SOFTWARE: THE GENIE IN THE BOTTLE by Tom Pittman

THE MEMORY PROBLEM by Ted Nelson

MEMORY, MEMORY, HOW MUCH MEMORY? by Stan Veit

APLomania by Eben F. Ostby

ROM

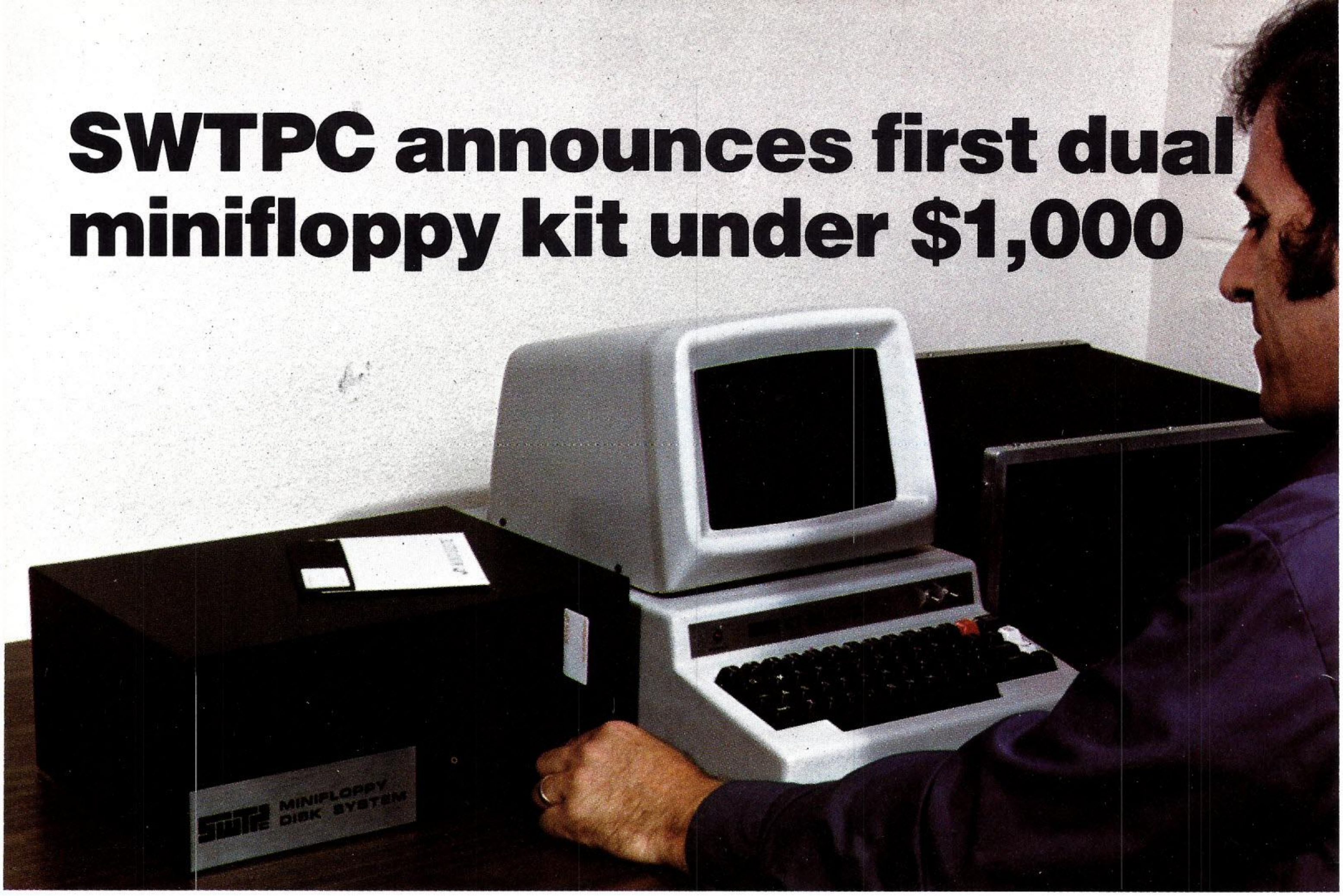
COMPUTER APPLICATIONS FOR LIVING

BINARY CLOCKS IN 0011/0100 TIME

Volume I, Number 2
August 1977 / $2.00

ROBERT GROSSMAN

SWTPC announces first dual minifloppy kit under $1,000



Now SWTPC offers complete best-buy computer system with $995 dual minifloppy, $500 video terminal/monitor, $395 4K computer.

SWTPC MINIFLOPPY DISK SYSTEM

$995 MF-68 Dual Minifloppy
You need dual drives to get full benefits from a minifloppy. So we waited to offer a floppy until we could give you a dependable dual system at the right price.

The MF-68 is a complete top-quality minifloppy for your SWTPC Computer. The kit has controller, chassis, cover, power supply, cables, assembly instructions, two highly reliable Shugart drives, and a diskette with the Floppy Disk Operating System (FDOS) and disk BASIC. (A floppy is no better than its operating system, and the MF-68 has one of the best available.) An optional $850 MF-6X kit expands the system to four drives.

$500 Terminal/Monitor
The CT-64 terminal kit offers these premium features: 64-character lines, upper/lower case letters, switchable control character printing, word highlighting, full cursor control, 110-1200 Baud serial interface, and many others. Separately the CT-64 is $325, the 12 MHz CT-VM monitor $175.

SWTPC 6800 COMPUTER SYSTEM

$395 4K 6800 Computer
The SWTPC 6800 comes complete with 4K memory, serial interface, power supply, chassis, famous Motorola MIKBUG® mini-operating system in read-only memory (ROM), and the most complete documentation with any computer kit. Our growing software library includes 4K and 8K BASIC (cassettes $4.95 and $9.95; paper tape $10.00 and $20.00). Extra memory, $100/4K or $250/8K.

Other SWTPC peripherals include $250 PR-40 Alphanumeric Line Printer (40 characters/line, 5 x 7 dot matrix, 75 line/minute speed, compatible with our 6800 computer and MITS/IMSAI); $79.50 AC-30 Cassette Interface System (writes/reads Kansas City standard tapes, controls two recorders, usable with other computers); and other peripherals now and to come.

SWTPC Southwest Technical Products Corp.
219 W. Rhapsody, San Antonio, Texas 78216
London: Southwest Technical Products Co., Ltd.
Tokyo: Southwest Technical Products Corp./Japan

Enclosed is:
_____ $1,990 for the full system shown above (MF-68 Minifloppy, CT-64 Terminal with CT-VM Monitor).
_____ $995 for the Dual Minifloppy
_____ $325 for the CT-64 Terminal
_____ $175 for the CT-VM Monitor
_____ $395 for the 4K 6800 Computer
_____ $250 for the PR-40 Line Printer
_____ $79.50 for AC-30 Cassette Interface
_____ Additional 4K memory boards at $100
_____ Additional 8K memory boards at $250
_____ Or BAC # __________ Exp. Date______
_____ Or MC # __________ Exp. Date______
Name__________ Address__________
City__________ State ______ Zip______

XIMEDIA PRESENTS

The SOROC IQ120

CURSOR CONTROL. Forespace, backspace, up, down, new line, return, home, tab, PLUS ABSOLUTE CURSOR ADDRESSING.

TRANSMISSION MODES. Conversation (half and full Duplex) PLUS BLOCK MODE – transmit a page at a time.

FIELD PROTECTION. Any part of the display can be "protected" to prevent overtyping. Protected fields are displayed at reduced intensity.

EDITING. Clear screen, typeover, absolute cursor addressing, erase to end of page, erase to end of line, erase to end of field.

DISPLAY FORMAT. 24 lines by 80 characters (1,920 characters).

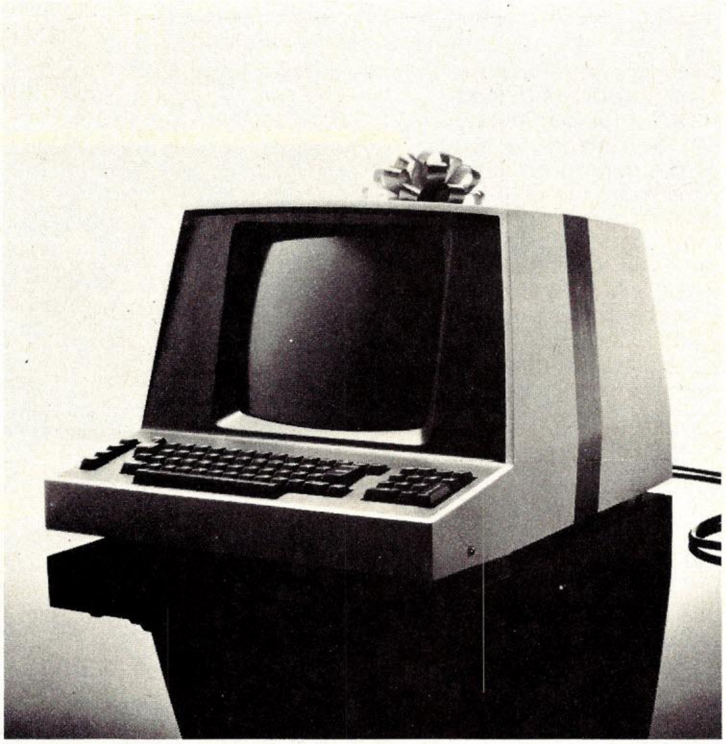
CHARACTER SET. 96 characters total. Upper and lower case ASCII.

KEYBOARD. 73 keys including numeric key pad.

REPEAT KEY. 15 cps repeat action.

DATA RATES. Thumbwheel selectable from 75 to 19,200 baud.

SCREEN. 12 inch rectangular CRT – P4 phosphor.



SPECIAL INTRODUCTORY PRICING

Kit $ 995.00 **Assembled $ 1,295.00**

(Price includes block mode, lower case and 24 line options.)

Specials of the Month

North Star **Micro Disk** with power supply and cabinet	Kit – $699 Assembled – $799
IMSAI I-8080 with **TDL** ZPU	Kit – $825 Assembled – $975
Digital Systems FDS Disk Drive with **CP/M** Software (assembled only)	Single – $1,750 Dual – $2,350
Mountain Hardware **PROROM**	Kit – $145 Assembled – $210
Vector Graphic 8K RAM	Kit – $235 Assembled – $285

NOW WE'RE TOLL FREE

800-227-4440

(in California, Hawaii, and Alaska, call collect: 415/566-7472)

XIMEDIA OFFERS A FULL RANGE OF PRODUCTS FOR THE PERSONAL COMPUTER ENTHUSIAST AND THE SMALL SYSTEM DESIGNER. LET US QUOTE ON ALL YOUR HARDWARE AND SOFTWARE NEEDS.

OUR RETAIL STORE – THE COMPUTERIST™ – IS NOW OPEN IN SAN FRANCISCO. CALL US FOR DIRECTIONS.

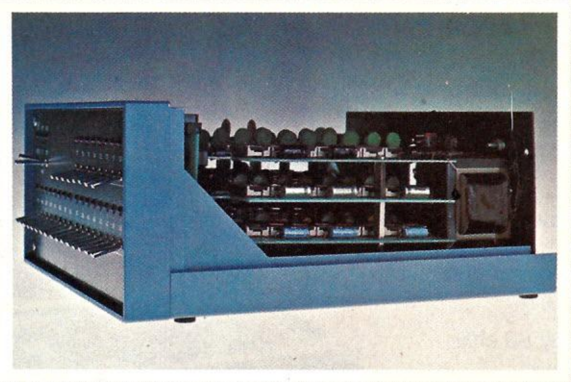
XIMEDIA

1290 24th Avenue, San Francisco, CA 94122
(415) 566-7472

COD orders freight collect. Orders with payment shipped prepaid. California residents add 6% sales tax. Please allow 3 weeks for delivery.

Powerful in computing muscle, yet small in physical size, the Altair™680b offers many special features at an affordable price. Based on the 6800 microprocessor, the 680b comes with 1K of static RAM, Serial I/O port, PROM monitor and provisions for 1K of PROM as standard components. It's good thinking, when you're interested in making a modest investment on a highly reliable computer, to consider the Altair 680b.

Our PROM monitor eliminates the necessity for toggling front panel switches to load bootstraps or manipulate memory contents. Only a terminal and programming language are required for complete system operation. With Altair System software—Altair 680 BASIC, assembler and text editor—you may begin problem solving immediately with ease.



By adding the 680b-MB Expander card, many options are currently available:

*16K Static Memory Board—Increase your system memory with 16K bytes of fast access (215 ns), low power (5 watts per board) static RAM. 680 BASIC and assembler/text editor are included free with purchase.

*Process Control Interface—A PC card that uses optically isolated inputs and relay outputs that transmit sensory information to and control signals from the computer. A diverse world of control applications is opened up with the Altair 680b-PCI.

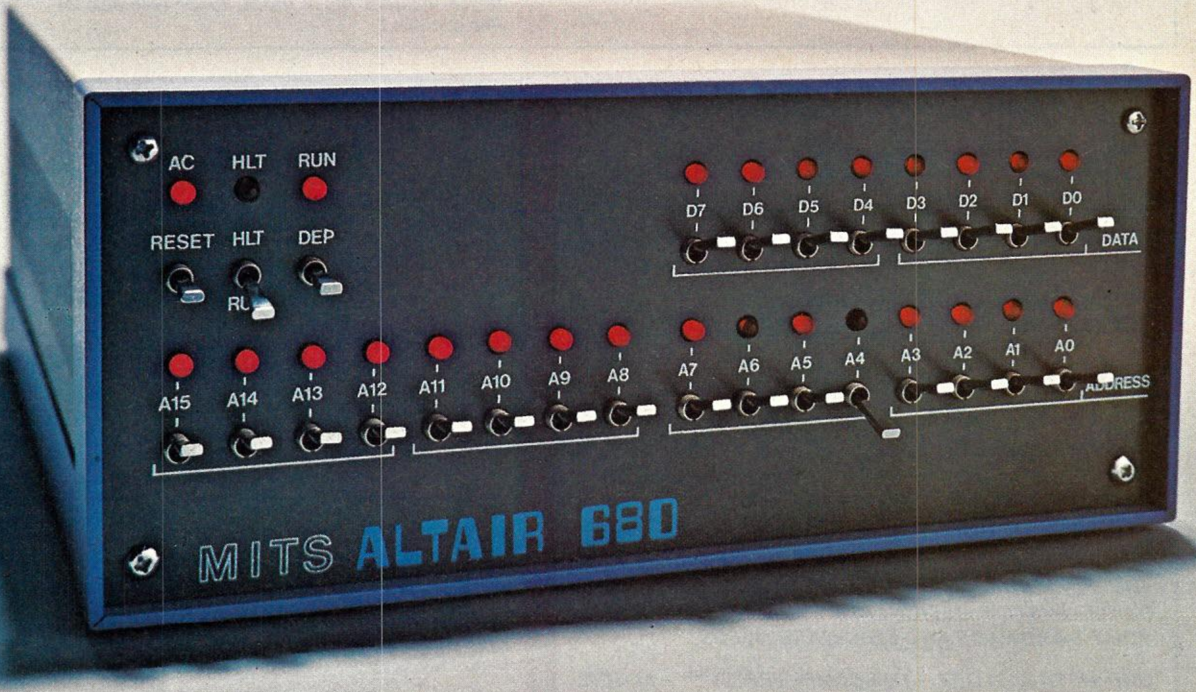
*Universal Input/Output Board—If your I/O needs exceed the serial port already on the main board, augment your I/O channels with the 680b-UI/O. By implementing the optional serial port and two parallel ports, you can simultaneously interface to four terminals.

*New Addition—Kansas City Audio Cassette Interface—Use the 680b-KCACR to interface your Altair 680b with an audio cassette recorder for inexpensive mass storage of programming languages, programs and data.

Available in either full front panel or turnkey models, the Altair 680b presents many computing capabilities at a low cost—without skimping on performance. See it today at your local Altair Computer Center or contact the factory for further details.

mits

Good Thinking.



2450 Alamo S.E. Albuquerque, New Mexico 87106
dealer inquiries invited.

Volume I, Number 2

Contents

August 1977

FEATURES

17 **Memories Are Made of This** by Lee Felsenstein
An introduction to the fundamentals of computer memory, including a previously undiscovered memory element developed by Free-Fall Ferris.

20 **Memory, Memory, How Much Memory?** by Stan Veit
In which an attempt is made to answer a question as age-old as the computer itself.

26 **Alice Through The Video Terminal or An Electronic Metaphysical Fantasy** by Sally Steinberg
A first timer's tale on discovering computers.

32 **Software—The Genie in The Bottle.** by Tom Pittman
The grand old master of Tiny BASIC takes you tripping through language land.

40 **Your Computer or Your Wife** by Susan Gilpatrick
The view from the distaff side.

41 **The Kit And I** by Richard W. Langer
One on one with a computer kit, by someone who's never even soldered before.

46 **Tooling Up** by Frank Becker
Tips for the do-it-yourself hardware beginner.

55 **Binary Clocks** by Caxton C. Foster
What could be more appropriate in this era of computers than a hand-crafted wooden clock—with a binary display?

58 **Computer Power and Where It Comes From** by Joseph Weizenbaum
Gamesmanship and how the computer works. Or checkmating the user.

76 **A Day in the Life of Morsus Computer Taberna #13** by Paul C. Conover
Thinking about cashing in on the computer craze by opening your own store? Better read this first to check out what you're getting into.

80 **Forget It!** A short story by Henry Melton
What if your computer couldn't forget it?

86 **APLomania** by Eben F. Ostby
With more available memory than ever before, APL is beginning to make inroads on the personal computer. Is it a language you should consider for home or small business use?

95 **The Brain Robbers** by Hesh Wiener
Honesty may be the best policy but. . . .

COLUMNS

10 **Missionary Position by Theodor Nelson**
Reflections on "The Memory Problem."

12 **The Human Factor by Andrew Singer**
What promise does software hold—for man?

14 **Legal ROMifications by Peter Feilbogen**
When is your computer tax deductible?

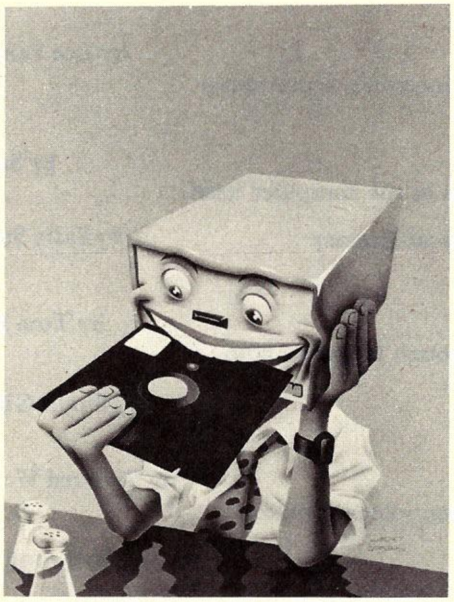
99 **FutuROMa by Bill Etra**
Memories to come.

DEPARTMENTS

4 **On the Bus**
8 **Reader Interrupt**
11 **Eve 'n' Parity**
49 **Run On Micros**
50 **Centerfold**
93 **Babbage and Lovelace**
93 **The Noisy Channel**
98 **Advertisers' Index**
100 **PROMpuzzle**

ROM is published monthly by ROM Publications Corporation, Route 97, Hampton, CT 06247 (Tel. 203-455-9591). *Copyright © 1977 by ROM Publications Corporation. All rights reserved.* Reproduction in any form or by any means of any portion of this periodical without the written consent of the publisher is strictly prohibited. The following trademarks are pending: floppyROM, futuROMa, The Human Factor, Legal ROMifications, Missionary Position, Noisy Channel, On the Bus, Reader Interrupt, ROMdisk, ROMshelf, ROMtutorial, Run On Micros, and PROMpuzzle. Opinions expressed by authors are not necessarily those of *ROM* magazine, its editors, staff, or employees. No warranties or guarantees explicit or implied are intended by publication. Application to mail at second-class postage rate pending at Hampton, CT 06247. Membership in Audit Bureau of Circulation pending.

On the Bus



Robert Grossman with, as *Graphis* put it, "the subtle power to change the way we perceive reality," is one of the world's outstanding cover artists. His work has graced just about every major periodical—*Time, Newsweek, New York, Esquire,* the London *Times, The New York Times,* and *Think,* to name only a few. Participating in numerous shows, his dazzling airbrush display has been honored with a long string of awards from groups such as the AIGA, the New York Art Directors' Club, and the Advertising Club of New York. Presently a cartoonist for *Rolling Stone,* Mr. Grossman feels that satire is one of the most effective ways of reaching his audience.

Hooked on crossword puzzles at an early age, **Daniel Alber** now constructs as well as solves them. Part of the Brownstone Renovation generation in New York, when he's not constructing puzzles for the likes of *Field and Stream, The New York Times,* and *ROM,* he's reconstructing olden golden rooms in his Brooklyn based house.

Frank Becker was introduced to electronics kit-building about two or three years ago by his brother. He finds that his interest provides "some peace and relaxation" after his long days as an excavating contractor—a business he launched about four years ago and which, as yet, is not computerized.

Paul C. Conover of Oakland, California is a successful consultant to the retail computer industry and the problems elucidated in "A Day in The Life of Morsus-Computer Taberna #13" are all from experience. Only the names have been changed to protect.... Mr. Conover was affiliated with the Altair Computer Center program for MITS and has been a principal in a computer peripherals company in California. He spoke at the National Computer Conference in Dallas in June on the subject "Starting A Retail Computer Store."

Bill Etra is an Instructor of Home Computing at the New School for Social Research in New York. He is co-inventor of the Rutt/Etra Video Synthesizer—the first portable voltage control analog video synthesizer, as well as the Videolab. His main interest is videographics, and many of his works have appeared as cover illustrations on various periodicals and books including *Computers in Society* and *Broadcast Management and Engineering*. His current research centers on "The Computer as a Compositional Tool for Video."

Peter Feilbogen attended the Rutgers School of Business Administration and Brooklyn Law School. In addition to being an attorney, he is also a Certified Public Accountant. He has been a sole practitioner on Long Island for approximately ten years, and treasurer of Data Information Services, Inc. An avid tennis player, he is currently trying to improve his game with the aid of a computer.

Lee Felsenstein was born in Philadelphia and grew up wanting to be an inventor. Outside of that, he bears no resemblance to W.C. Fields whatsoever. Instrumental in establishing the first experimental public-access information-exchange system in 1972, he is presently engaged in further development in that area of communications. In his spare time he has designed the Pennywhistle 103 modem, the VDM-1 video display module, the Sol terminal/computer, and the VID-80 video display card. Lee was also instrumental in forming the original Homebrew Computer Club and currently serves as its "toastmaster."

When he was ten years old, **Caxton Foster**'s grandfather gave him an old Viennese clock which chimed the quarter hours. Ever since then he has been fascinated by the display of time passing and is now an avid clock-auction attendee. Thinking it would be interesting to devise a horological instrument that, if it somehow dropped into a time warp and ended up in the seventh century, could be easily reproduced and explained by the technology of the times, he ended up with the binary clock. The electric motor, which strayed from the original plans, can be replaced by a pendulum about a meter long.

Susan Gilpatrick is a housewife turned kit-builder. She was going along her merry way, trying to finish college and stay out of trouble, when her husband George decided he wanted to have his own computer. He assembled it from a kit and, knowing what he was doing, didn't need more than two weeks to straighten out all his errors. Not to be outdone, Susan built one as well. Not knowing what she was doing, the computer ran perfectly as soon as it was plugged in.

Richard W. Langer is a free lance writer whose articles have appeared in such diverse magazines as *New York, Family Circle, House and Garden,* and *Esquire.* Currently he is a columnist with *The New York Times* and at work on his fifth book.

Henry Melton is a member of Science Fiction Writers of America who got his B.S. in Physics from the University of Texas. He started working with computers by building a SC/MP based text-editing system. "I had to—you should have seen my typing." Unlike the hero of "Forget It," his story in this issue, Melton has a wife who is sympathetic about both his writing and his work with computers. He is currently a broadcast engineer for KLBJ radio as well as a free lance writer.

Theodor Nelson is the author of the classic *Computer Lib/Dream Machines,* a Whole Earth style catalogue of computer machinations. Presently at the Department of Mathematics of Swarthmore College where he is working on the Hypertext Project, Ted specializes in highly interactive systems for graphics and text. His past experience includes a stint at Dr. Lilly's Dolphin Laboratory and work as a consultant for Bell Lab's ABM system.

Eben F. Ostby first began experimenting with computers while at the Pomfret School, which had a PDP-8. He went on to become the first Computer Science major at Vassar College, from which he recently graduated with honors. Ostby has done extensive work with graphics in APL, and recently programmed what he thinks may be the first cartographic project in APL.

Tom Pittman 's dream, ever since high school, was to own a computer. Well, now he does—in fact he owns several. Starting with an Intel prototype 4004 system in 1972, he has branched out, turning his hobby into a business, albeit a tiny one, as the main implementor of Tiny BASIC.

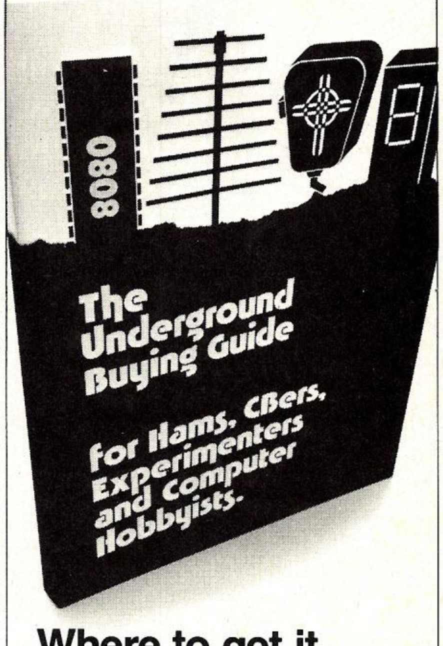
Andrew Singer has been hooked on computers since he first built one in 1958. A hard/software consultant, he is fluent in thirty computer languages and knows more than enough about twenty species of machine. His work has included the first medical information retrieval system based on ordinary clinical records, and a large and intricate system for interactive selection of data from public opinion polls. He believes that most software is poorly designed and unspeakably rude, and his Ph.D. research is aimed at improving the architecture and human engineering of interactive systems.

Sally Steinberg is a free lance writer based in Cambridge, Mass., "just like everybody else." She has been a contributor to the *New England Magazine* of *The Boston Globe,* and to *Boston* magazine, as well as several other publications. Her household contains a calculator but not a computer "yet," she says, "but now the whole family likes playing with computers." To see how they found out about them, turn to "Alice Through the Video Terminal."

Stan Veit has been in the computer field for about twenty years, originally as a technical writer and instructor of computer maintenance. He is co-author (with Leslie Solomon, Technical Editor of *Popular Electronics*) of *Getting Involved With Your Own Computer.* Veit founded the first computer store on the east coast.

Joseph Weizenbaum is Professor of Computer Science at the Massachusetts Institute of Technology. He is best known to his colleagues as the composer of SLIP, a list-processing computer language, and for ELIZA, a natural-language processing system. More recently, he has directed his attention to the impact of science and technology—and of the computer in particular—on society.

Hesh Wiener is the editor of *Computer Decisions,* the most widely read computer trade publication in the United States. Previously he was on the academic staff of the University of California at Berkeley, where he headed the Computer Education Project at the Lawrence Hall of Science. During that time he spent a year teaching blind children to use computers. He designed and built some of the equipment used by the blind students.



8080
The Underground Buying Guide
for Hams, CBers, Experimenters and Computer Hobbyists.

Where to get it.

Equipment, parts, supplies and services. Hard to find and standard items at bargain prices.

Over 600 places to find transceivers, antennas, surplus, new and used equipment, μPs/computers, ICs, components, assortments, assemblies, discounted items, test equipment, peripherals, etc. Hundreds of large and small mail order sources.

A complete directory divided by sources, items and locations. Saves countless hours of shopping. Easily pays for itself through comparative buying. Contains no advertising.

Rush my order. I enclose $5.95 plus 55¢ postage and handling. Californians add 39¢ sales tax. Full refund if not completely satisfied within 10 days.

Name ______________________

Address ______________________

City/State/Zip ______________________

Primary interest: Amateur Radio ☐ CB ☐ Experimenting ☐ μPs/Computers ☐

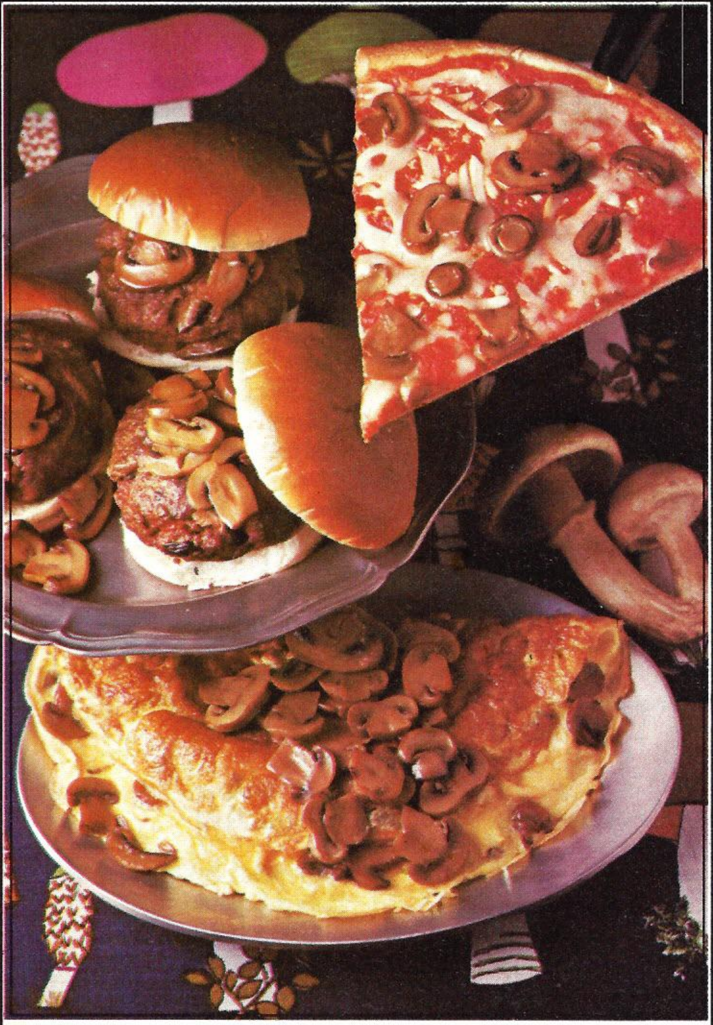
Send to: Peninsula Marketing
Dept. P
12625 Lido Way
Saratoga, CA 95070

COLLIER IS THE BEST ENGRAVER, PRINTER IN THE COUNTRY.

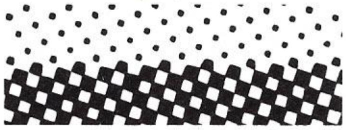
Because. A revolution in engraving has happened. Collier split the dot. And because of the new Laser Beam, Collier is now light years ahead of the rest. The laser does it. Here's how it works. Technically, the laser beam which is used to control film exposure (thus "Program" the engraving) is split into six sectional beams. Each of these is digitally modulated by computer to transfer half-dots of picture information per scanner revolution. Since two picture-information "bits" are available per dot in circumferential direction, it's possible to expose a smaller area in the second "orbit"...or even completely omit it (thus producing an actual half-dot or even an elliptical mini-dot). If that seems to all sound a lot like gobbledygook, don't worry about it. The important thing is, it's here and it does work and it gives your image resolution that you never could get before. We'll be happy to demonstrate it for you. Even happier to produce your next set of four-color letterpress, offset and gravure engravings. Call Collier Graphic Service Company Inc., 240 West 40th Street, New York, New York 10018/(212) 840-0440.

ATLANTA	BOSTON	DETROIT	HARTFORD	NEW YORK	PHILADELPHIA
(404) 892-2383	(617) 965-5660	(313) 259-2111	(203) 367-0706	(212) 840-0440	(215) 988-0110

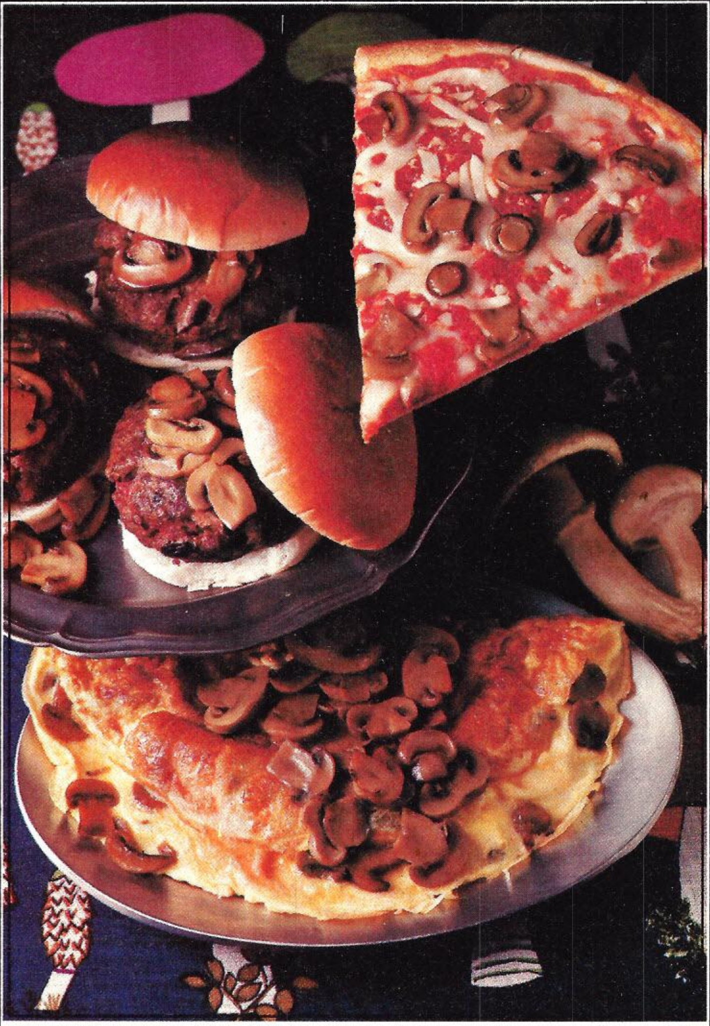
OR CALL TOLL FREE (800) 221-2585/(800) 221-2586



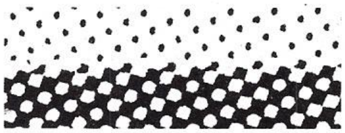
THE ABOVE ENGRAVING WAS MADE WITH THE CONVENTIONAL PLATE MAKING PROCEDURES, AS YOU CAN SEE, IT LACKS COLOR FIDELITY AND SHARPNESS, IF YOU COMPARE IT WITH...



THE CONVENTIONAL ENGRAVING DOT.



THE COLLIER'S LASER BEAM ENGRAVING, AS YOU CAN SEE FROM THE ABOVE, HAS BRILLIANCE, SHARPNESS AND COLOR FIDELITY THAT ONLY COLLIER'S DOT CAN PROVIDE.



THE COLLIER LASER DOT.

THE COLLIER ADVANTAGE. COLLIER GRAPHIC SERVICES COMPANY, INC., 240 WEST 40TH STREET, NEW YORK, NEW YORK 10018

Reader Interrupt

Dear ROM,

A crossword puzzle in a computer magazine? You gotta be kidding. Cut the fillers and put in some meat.

Ray Erpf
Akron, Ohio

Dear ROM,

Enjoyed the crossword puzzle. Since I'm a beginner, I knew more of the English than the computerese, but I learned as I went.

Frank Reeves
Santa Fe, New Mexico

Dear ROM,

Gordon Morrison's "Altair and the Art of Motorcycle Shop Maintenance" was great. But couldn't you make the articles more general? I'm in the plumbing business and although there's a certain similarity as far as inventory goes, there isn't enough for someone like myself who's totally unfamiliar with computers to go by. The ROMtutorials are great, by the way.

Chris Grumwald
Newfane, Vermont

Dear ROM,

OK, you SOCked it to me with "Telegrasping at Midnight." But where can I get my own SOC? I can think of a million applications for SOCs but I never heard of them till now, and my local Radio Shack says they don't know anything about them.

Philip Nobile
La Jolla, California

Just keep reading ROM. *Avery Johnson is busy putting together a homebrew unit for your pleasure. As soon as he's through knitting, we'll run the pattern.*

ROM

Dear ROM,

Where can I get DREADCO's address? I've developed a self-orienting bread which, when covered with either spread-plus or spread-minus (both of the low-priced variety) is guaranteed to land buttered side down or buttered side up respectively. Should be ideal for disorienting the opposition at political picnics.

Carl Hargrave
St. Paul, Minnesota

Your project sounds a bit slippery. However, if you wish to pursue DREADCO on the matter, you should be able to reach them c/o New Scientist, *King Reach Tower, Stamford St., London SE1 9LS, England.*

ROM

Dear ROM,

The thing about "A Chip Is Born" is I'd been working with chips for several years now, and never thought to ask how they were manufactured. Found the whole thing fascinating. How about one on magnetic tape?

Roxanne Albert
Birmingham, Alabama

Glad you asked. It's coming up.

ROM

Dear ROM,

And here I thought the walnut sides on Sol were a great marketing coup.

Stuart McPherson
Des Moines, Iowa

So. You don't call recycling scrap lumber at less than wholesale prices a marketing coup?

ROM

Dear ROM,

I was very glad to see, if you'll pardon the unintentional pun, Hesh Wiener's piece on the Braille terminal. (Actually, the unintentional pun doesn't even bother me the way it might have before I read the article.) But how about an inexpensive terminal where everything doesn't come out backwards? Would be a lot more convenient.

Irene Cabrowitz
Brookline, Massachusetts

All right, how about it? Write us an article when you finish the project.

ROM

Dear ROM,

Ted Nelson's column alone is worth the price of a subscription. Here's mine. No relation by the way.

Everet Nelson
San Francisco, California

And for three years at that! Sure you're not a relative?

ROM

Editor and Publisher
Erik Sandberg-Diment

West Coast Editor
Lee Felsenstein

Assistant Editor
Janet C. Robertson

Contributing Editors
Sandra Faye Carroll
Bill Etra
Louise Etra
Ed Hershberger
Avery Johnson
Richard W. Langer
Theodor Nelson
Robert Osband
Frederik Pohl
Andrew Singer
Alvin Toffler

Crossword Puzzle Editor
Daniel Alber

Art Director
Susan Reid

Contributing Artists
Robert Grossman
Cindy Hain
Luis Jimenez
Korkie
David Macaulay
Linda Smythe

Staff Photographer
Thomas Hall

Composition
Lynn Archer

Editorial Assistant
Donna Parson

Special Assistants
Jennifer L. Burr
Joanne Zeiger

Counsel
Peter Feilbogen

A Note to Contributors:
ROM is always looking for good computer applications articles from people with up-and-running systems. We also will be glad to consider for possible publication manuscripts, drawings, and photographs on other computer-related subjects. Manuscripts should be typewritten double-spaced, and a stamped self-addressed envelope of the appropriate size should accompany each unsolicited submission. Although we cannot assume responsibility for loss or damage, all material will be treated with care while in our hands. Contributions should be sent to ROM Publications Corporation, Rte. 97, Hampton, CT 06247.

You're curious enough to read ROM now, you'll be curious enough to read ROM every month. So subscribe!

SAVE $ 7.00 over the single-copy price with a one-year subscription

SAVE $20.00 over the single-copy price with a two-year subscription

SAVE $33.00 over the single-copy price with a three-year subscription

SAVE EVEN MORE with the Rich Uncle Special:

For the first six months, and the first six months only, ROM is offering genuine inflation-proof lifetime subscriptions at $250.00 each. No matter what happens to the dollar compared to the yen, the mark, the franc, or even the yak, with the Rich Uncle Special you're assured of a lifetime supply of ROMs. At today's rate of inflation, a year's subscription may well be selling for that much in only a decade.

GUARANTEE :
If not satisfied with **ROM** at any time, let us know. We'll cancel your subscription and mail you a full refund on all copies still due you.

ROM
COMPUTER APPLICATIONS FOR LIVING

ROM Publications Corp.
Route 97
Hampton, CT 06247

Name ______________________

Address ______________________

City ______________________

State ____________ Zip ____________

☐One year $15 ☐Two years $28 ☐Three years $39
☐Check or money order enclosed. ☐Lifetime $250
☐Master Charge ☐BankAmericard
Exp. date ____________ Card# ____________

Please allow 4-6 weeks for delivery.

Missionary Position

THE MEMORY PROBLEM

by Theodor Nelson

People have a funny way of talking about the memory problem, as if it had a single simple solution that would be along any day now. "Soon there will be a disk—a new strange device—a better language—or a chip—that will solve The Memory Problem."

It'll never happen.

Basically, the memory problem is that there isn't enough fast memory, directly addressable, on your computer. (Or anybody else's.) Core memory, or whatever they're calling it these days, isn't quite big enough for the get-it-done-ers, isn't nearly big enough for the dreamers. This was true at the beginning and is now and will be later. This is true for your Altair and true for the PDP-10 down the block and true for the Illiac at Burpleson AFB. The memory problem is that there is never, *ever* enough. (In this it strongly resembles The Money Problem, as well as The Sex, Quiche, Time, and Energy Problems.)

Everybody has his own myth about the solution to the memory problem, if any.

Many solutions have been proposed. Obviously we want a generalized solution, meaning one that will work in the next state.

Some say the solution is more hard work. This has the advantage of always being possible and never requiring more equipment. It leads people to filling every spare bit and loose corner. It leads to programs which can never be modified. It leads to job security and bad programs.

A very popular interim solution may be stated more briefly. *More.* Especially, *more core.* This works temporarily for the get-it-done-ers who realistically only needed a little more. It is of no use for the dreamers. But soon even the former group will allow themselves to dream again, a little, and it's time for more core again. (The most parsimonious phrasing of the memory problem may be this: you never *can* get enough main memory. The faster you enlarge your computer, the sooner you need a bigger one.)

The hardware dinks—those who (at the upper level) think of software only as a stopgap, or who (at the lower level) think most programs fit on one page, and besides, what they like about home computers is the soldering, think—have always thought, always will think—that there will soon be some piece of equipment—some new kind of tape, or chip memory circuit, or radical new computer architecture—that will take care of the memory problem.

The classic such solution was to have been *associative memory:* special circuitry that would cause all data items with certain traits or connections to stand up and be counted. Supposedly this would take care of everything. Well, now there's a machine like that on the market—the Goodyear Staran—and it costs a great deal of money for a minicomputer-sized unit. Presumably it's a fine machine—but you need what we call a "well-tailored, special application" to justify it.

Now the Texas Instrument bubble chip has come along, and we hear the same sorts of things. 92K bits! For a hundred dollars! (Soon.) And fast, supposedly, though with new complications. Soon it will hook on the Altair and brethren. Is this the solution?

No.

A floppy disk, the disk itself, costs under ten dollars, and holds a quarter of a million characters that you may want permanently. If you put things permanently in the bubble chip, it will cost you a hundred. There's the problem.

Because programs ordinarily take place in addressable main core, the natural programming assumption is, "Get everything into addressable core." But there is the mistake. Main core can never be big enough. The distinction between core, disk, and tape is perpetual. There will *always* be core, disk, and tape. Not literally; I mean rather that there will always be some kind of fast, expensive internal memory, some kind of slower, cheaper mass memory, and some kind of *much* slower, cheaper mass memory.

This means that programs must always be *tiered:* some things must go fast and go on always, staying in core. Some things must come and go, from and to the slower regions. The management of fetching and storing is a fundamental aspect of computer programming. Doing this sort of thing right, or failing to do it right, constitutes much, perhaps most, of what goes on under the name of programming: getting and putting, backing up, updating, swapping, and overlaying, as well as the more advanced tribulations of access methods, cache and page fault optimization, allocation of overflow areas, and so on. Computer techniques are, in large measure, ways of handling this tiered memory: what's on slow-and-cheap memory is waiting for its chance to get into small-and-fast.

Professional programmers in the 1950s were excited to get into 8K of core. Now the day has come when 64K on your Imsai doesn't seem like quite enough.

Part of the problem is how much more we've come to expect. We want to chain our programs. We want to mix languages. We want to bring several data structures in at once. Now it's not enough to crunch a few numbers and print a few lines of text; we want to bring in TV camera images and put out whole orchestras via D-to-As. These things call for memory, memory, memory.

One very generalized solution to the memory problem is somehow to make all memory, fast and slow, behave as though it were one: set up a huge time-sharing complex wherein we do not notice any distinction between core and other forms of memory. Vast tracts of empty core stretch before the programmer, like the endless prairie that greeted the pioneers of the 1840s. (This is regrettably called "vitual memory," though it is in fact *virtual huge core.*) Of course this works only on time-sharing computers, where swapping is the name of the game and we might as well swap an individual's core by the acre while we're at it.

Such systems work; some work well. The thing is that they only work on a vast scale, and at big-company prices. So for those of us who need personal computing, they're no solution at all. (But at least it's awfully generalized: they've had operating systems of this type running each other as subprograms several deep. For some, recursion is its own reward.)

Another generalized solution is the back-end processor: a complete subsystem, attachable to your regular computer, that hands over data from within a grand information web as requested, shouldering all the file-management problems. That way the main computer just asks for a certain piece of something within the grand information web, just as if it were in some handy part of core, and lo! it appears. Such was Bachman's Integrated Data Store on the old General Electric computers; such are the minicomputer-based information systems now being backed up to 370s. I think it likely that such back-end storage and retrieval will soon appear as a telephone service for personal computing, necessarily at a far lower cost than conventional time-sharing. Such, I think, is also the solution to the hypertext problem: you can only put huge libraries on shared computer networks if you delegate the file management to very specialized retrieval subsystems.

But still there is no general solution to the memory problem-and there will be none. We will just get more core and more mass storage, and hope for cleaner and cleaner ways to interrelate them.

If hardware does not provide a "solution" to the memory problem, at least it keeps getting better and cheaper. Little stuff for the personal market is all over. Ten years ago I don't think there were any disks that cost under fifteen grand; now a dual floppy is around $2500.

The emerging personal computer market calls for a lot of similar redesign: for years they've been designing mass magnetic memories only to be used in big installations. Tapes, for instance. Computer tape drives were originally designed only for big-computer styles of application: you could only add things at the end, not replace in the middle. Then came the LINCtape, which was ahead of its time in 1962. It was developed at Lincoln Laboratory for the LINC personal computer, and it's still around; the one in my livingroom works very nicely. (Computer Operations makes it now; Digital Equipment's version is called a DEC tape.)

Same for disks: they were overdesigned. The old expensive disks all went thirty revolutions-per-second or faster; much later it was noticed that disks didn't have to be rigid; five revolutions-per-second was adequate for many purposes, thank you.

Same for the magnetic card memories. The RACE and CRAM of the sixties, for instance, had boxes of long vinyl strips the size of your arm, with magnetic recording on them. Fingers would select (by notches, as on a Linotype) the right strip; then the strip would shoot around a drum, divulge (or revise) its contents, and drop back into the box.

But what of the lowly Hollerith-size mag card, as on some typing devices? Now there's a recording medium, at about one dollar a card and 64K bits, that's about right for dinky computers—how about it?

Vinyl-iron technology is by no means exhausted. How about a megafloppy? You could drive a free-flying read/write head with compressed air and zip it around a sheet of any size. Perhaps card-table size would be about right; you could roll up the sheet to change contents.

There are openings for other weird things that exist but never quite have gotten around. For instance, sprocketed 16mm mag tape is used in the movie industry and on IBM's Mag Tape Selectric—does anybody have drives for that?

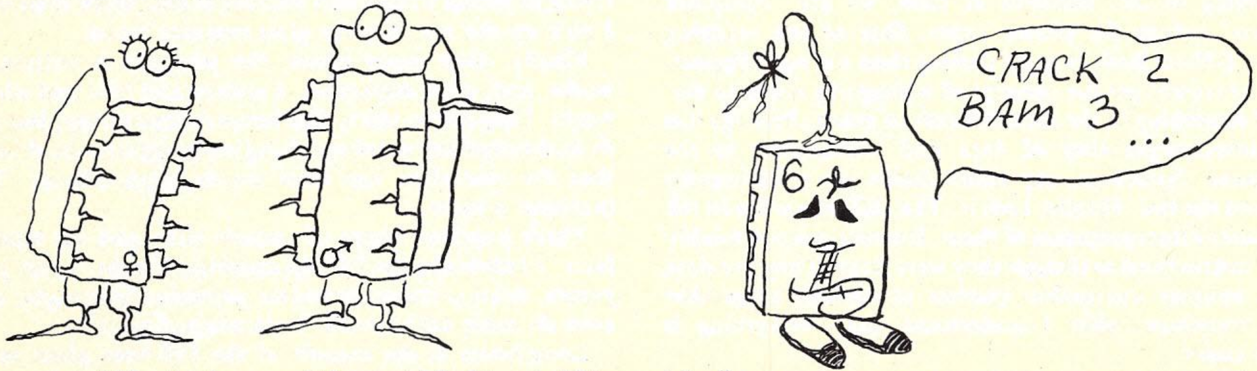
Personally, I'm still hankering after the floppy disk juke box somebody was offering last year, but the literature I sent for never came—are they still funct? (Then there was the "rotor memory" somebody advertised in *Datamation* in the early sixties. It appeared to be a polyethylene mixing-bowl covered with magnetic oxide. The literature on *that* never came either.)

Oh, well. Gadgets come and go; there are two sizes of floppies, two sizes of Philips cassettes, two sizes of 3M cartridge. But beyond this natter, there are definite trends.

Some things change and some don't. Soon you can have the whole book, *War and Peace,* in core; in a decade you will be able to have the whole *movie* of *War and Peace* in core, a million dots per frame. But by then you'll want more. Today's largest PDP-20 will seem small for a home computer not very long from now.

We may expect to see, heh heh, some dramatic changes in store. But the relation between levels of store is not likely to change much. ▼

EVE 'N' PARITY



"Oh, that's one of those old-fashioned Chinese chips."

Cartoon by Korkie

The Human Factor

PROMISES, PROMISES

by Andrew Singer

reassembled by Dede Ely

As I reach back into the archives of my mind, I find a time of promise. It is a night in the early spring of 1961. I am seated at what might be the control console of the *Starship Enterprise.* The air around me is cold and dry. Overhead, the flat, blue gray light of technology flickers. A geometric array of lights and switches confronts me; at my fingertips is a small keyboard. At my right is a teleprinter; in the distance loom the machines of thought, logic, and communication. Banked like soldiers, they are at my command.

The computer itself is silent. Its thoughts, like ours, are an almost mystical flow of electrons. It sits surrounded by the tentacles of peripheral machines, its links to the world. Like eyes, ears, and mouths, they clatter louder than a thousand tongues speaking simultaneously. The numerous whirs, clicks, clanks, and clatters just penetrate the dull roar of twenty tons of air conditioning and the whine of two hundred kilowatts of electricity. It is hard to hear myself think. Like Niagara Falls, the noise separates me from the outside world with a curtain of sound. When I leave the room, I will be shocked by the sudden silence outside.

But I am not in outer space; I am down to earth in Cleveland, Ohio in 1961. The address is the Case Institute of Technology, and I am seated at a Burroughs Datatron 220, a state-of-the-art computer.

This is the first time I have been alone in this room. A patient team of computer types has just spent fifteen grim hours trying to convey the most primitive principles of programming to me. Students at Case, we are "computer hackers" before the phrase exists. One of the teaching group, Gilbert Steil, is more patient than the rest. Repeatedly, he tries to get the concept of a program across to me. I find something in the idea difficult to grasp. Perhaps it is the interchangeability of data and instructions in the computer. Somehow, my mind insists on a dichotomy between the two. Finally, I see it. The difference lies in the computer's interpretation of them. Instructions can modify other instructions as though they were data. They *are* data when another instruction chooses to identify them that way. Somehow, after I understand that, everything is much easier.

It is no accident that Gilbert is the agent of this discovery. He is co-author of a major assembler (a program for language translating) and he is an ace programmer. It was his idea that I visit the computing center and look at the machine. I suspect that he had spotted me for a potential "computer junkie" months before when we first became friends. I think he knew I would be hooked the moment I saw the computer. I don't think, though, he had expected a fifteen-hour struggle to land his fish. But, like most people who work with computers, Gilbert is tenacious.

That is how I have come to be sitting in front of the 220's console with a reference manual for the machine in my hand and a simple programming problem. For the first time I am on my own, a human struggling to master a computer which is completely at my disposal. If I can accomplish this, I will have access to an immense amount of power. Although this power is abstract, almost intellectual, its trappings are tangible. When I sit alone at the console, I have a million dollars' worth of hardware under my thumb. I can see it. I can hear it. I can feel it.

Despite its huge size and impressive appearance, the 220 is a small computer. It can store only fifty-five thousand decimal digits. The largest program that can fit in memory is only five thousand instructions. Although the tasks it can perform are powerful even by the standards of the future, the 220 is slow, requiring ten thousandths of a second for even the simplest instructions. Top speed for the printer is only 120 lines per minute. The computer is built with thousands of miniature vacuum tubes and miles of cable. But in 1961, it is the state-of-the-art in commercial computers, and I can feel that too.

Sitting there, I grapple with my simple problem for hours. Reading the manual, looking at the console, pressing keys, switches, and buttons, watching the effects of displays and indicators, I explore the maze which is the machine. From time to time I am interrupted, as someone with higher priority (virtually anyone has higher priority than I do) "bumps" me from the machine. The sense of loss I feel is painful. These people are taking away *my* computer. The ease with which they use the machine is maddening. Most of them produce fat listings with thousands of instructions. My program's ten instructions don't even fill one page.

During these breaks, I leave the machine room and sit in the hallway outside, a little lobby furnished with a leather couch, an end table, and a lamp. The couch sits beneath what is to become a tired cliche in computing centers, but which seems funny at the time. It is a glass case containing an abacus. The legend reads, "In Case of Emergency, Break Glass!" Head aching, gulping coffee and donuts, I sit under the abacus, hunched over my reference manual, trying to pound its contents into my brain. More than once I pick up the axe, but the glass remains intact.

Finally, after many hours, the program is written, it works, and, most important, I understand how and why it works. I have been thirty-five hours without sleep, but as I sit in the machine room watching my program run, I know that the marathon has been worth every minute. The machine is mine.

Three hundred hours of console time and six months later, I can make the 220 do anything I want. I can play games, draw pictures, translate primitive languages, and even do some useful work. I am meshed with it.

Long hours at the console of the 220 have given me a unique appreciation for the thoughtful human engineering that has gone into its design. The console enables a remarkable symbiosis between me and the machine. In 1961 there is real concern for the human operator of the computer. This concern is based on several premises. Program testing, or "debugging" in jargon, is regarded as an important and difficult aspect of program development. Most users of the computer are scientists and other people of importance. Although the machine's time is extremely valuable, the user's time is also considered significant. The author of a program usually debugs it and often runs it as well. Since only one person can use the computer at a time, everyone benefits if program debugging can be accomplished as quickly as possible. Furthermore, as the computer is a relatively new device, most users will be new to it and thus there is a value in making it directly comprehensible. The console serves this purpose by *presenting* the computer to the user.

A few machines, successors to the 220, will even improve upon its console, but just a few years after 1961 all of this will be gone.

Scanning my memory, moving forward in time, it is 1965. . . . I am a consultant to a professor of English at New York University. He wants to use the NYU computer to construct an annotated edition of Henry James's *Daisy Miller.* He has the original manuscript and the five successive revisions James made, all of which have been punched on paper tape in an obscure code used by some type-setting machine. James made each of these six revisions progressively longer, despite the many deletions in each revision. A quantum jump occurs at the third revision when he started using a dictating machine. The final version is nearly twice as long as the original.

In any event, the English professor believes it should be a simple matter to get the computer to compare versions and annotate the changes. Garnering computer facilities for the project is no problem. English departments are not heavy users of computers in 1965, and university computing centers are always on the lookout for new ways to justify their own existence. When it learns of the project, the NYU Computing Center is anxious to help. Funding the programmers for the project is even less of an obstacle. Like computing centers, granting agencies are always interested in new and unusual undertakings. Coupling the computer with a project of artistic and scholarly merit in the humanities yields a natural grant-winner. A colleague of mine is invited to work on the programming and he, in turn, invites me.

And that is why for the past two weeks, I have been studying a wonderful set of IBM manuals which describe NYU's 7094 computer and its operating system, affectionately named IBJOB.

Things change quickly in the computer field. By 1965, most new computers are relatively large, and considerably faster than the 220. In fact, they are so fast that punched-card input and printed output devices cannot keep up with their computational and data-processing capability. Neither can humans. It is said to be uneconomical to tie up these large machines with card reading or printing. Instead, new small satellite computers read cards and write them on magnetic tapes which are then carried over to the central computer for processing. It, in turn, writes its output onto other magnetic tapes that are carried back to the satellite computer which then prints their content. IBM is adept at setting up these off-line input/output installations.

It is, similarly, no longer regarded as economical to allow people to operate the computer for their own programs. Instead, large programs called "operating systems" have been devised which actually run the computer. Human operators still have a supervisory role, and they must, of course, perform the menial tasks such as mounting tapes, but their reaction time is too slow for the really fast decisions which now must be made by the computer itself.

Thus far in my own career, I have successfully managed to avoid being a computer user in an environment dominated by an operating system although I have participated in the development of several. But, as I am beginning to discover, it is a very different thing to be a user. The operating system, IBJOB, is bad enough, but the manuals are even worse. Complex, tricky, and idiosyncratic as IBJOB is, the manuals cannot really describe it accurately. Or perhaps it would be more precise to say that the description is incomplete. Inevitably, at some crucial point in a manual, the information one needs is missing. Occasionally there is someone, an old hand at the system, who can supply the answer from experience. Often, however, the only way to find out is by trial and error. And what a process that is!

To utilize the machine efficiently, programs are "batched" so that the operating system may optimize use of the computer's resources. The effect of this batching from the user's point of view is that one's program is not run immediately. In fact, it may take as long as two or three hours to have it run. This turnaround time is very frustrating to the user. In the future it will be even more so. By 1971, most users of large batch operating systems will consider themselves lucky to get ten-hour turnaround. This delay provides one with plenty of time to do something else. The only problem is that what the user is really itching to do is to see how his program is doing.

On top of the tension caused by waiting for turnaround, the operating system forces one to test programs by a kind of remote control. Instead of a beautiful console and a machine dedicated to one's task, there is a new kind of computer language which is used to tell the operating system how to run the program. Meanwhile, the operating system itself is running a production line of programs. Even the terminology has changed. A program to be tested or run is submitted as a "job" for the operating system. Programming errors, bugs, which used to be discovered interactively in a matter of minutes at a console, take hours or days with the "help" of the operating system.

All of this becomes increasingly evident to me as work progresses on the text-comparison project. My morale drops lower than my productivity. By 1970, I will have worked in so many awful batch environments that nothing will faze me anymore, but in 1965 the shock is too great. After struggling several months to complete two small modules of the text-comparison system, which should have taken two weeks, I resign.

As I write these recollections in 1977, I still do not know if Henry James got his due from the IBM 7094. But it would not surprise me to learn that he did not. Many a programming project enthusiastically begun has vanished into oblivion. Hostile operating systems like IBJOB or its pernicious successor, OS/360, have not been the only or even the primary reason for this, but they have certainly been a contributing factor. Confronted with overwhelming odds, humans occasionally have the sense to retreat. ▼

Legal ROMifications

HOBBY OR BUSINESS?

by Peter Feilbogen, attorney at law

While April 15th has long since been ripped off the calendar, the pain lingers on. You may remember how you searched for deductions. This month let's look into whether you have a hobby or a business, an income or a loss.

Many people spend large sums of money on matters which interest them. On occasion some people are rewarded with substantial economic gains, others incur economic losses. One of your uncles, named Sam, is quite interested in such activities, so interested that he will argue vehemently with you about whether your interest is a hobby or a business, whether your losses are deductible, and how much of your gains are taxable.

Internal Revenue becomes interested in your pursuit when you realize a gain or when you deduct a loss on your tax return. You must become interested in your government's concern so that you may prepare the most favorable foundation for your contention. Obviously it becomes quite important to determine what the criteria are in determining whether an interest is a hobby or a business.

You may find it amusing to note that the Internal Revenue Code does not refer to the word "hobby" anywhere; instead you are engaged in "Activity Not Engaged In For Profit."

The hobby or business determination is quite complicated because, by law, any individual may pursue more than one trade or business simultaneously. To qualify as a trade or business, a pursuit must have as its prime objective a reasonable expectation of a profit; further, the individual must expend a substantial part of his business time in this pursuit.

Yet there is a form of relief available in the guise of a presumption written into the Internal Revenue Code. It states that if gross income is earned from an activity in *two or more years out of five consecutive years,* such activity is presumed to be for profit.

But wait, Internal Revenue Service can take back what Congress almost gave. If they do, it will mean that IRS may try to establish that such activity is not for profit.

How can you use this information? If your activity by Internal Revenue Code definition is a *hobby,* you may deduct expenses relating to your hobby to the extent that you receive income. Thus, a hobby can earn taxable income, or break even, but you may not deduct losses on your tax return.

If your activity falls into the category of a *business,* and if it has shown a loss, you may deduct such a loss. Of course a gain must be reported.

You must be prepared to support your contention that you have a business. The following are some suggestions:

- Hold yourself out as being in business by letting your friends, neighbors, and business associates know you have an enterprise, e.g., business cards or a sign in front of your house.
- If your jurisdiction requires filing the names of businesses, have you filed such papers?
- Obtain a Federal Employer Identification number even if you do not pay salaries.
- Keep financial records, even if informal.
- Maintain a separate bank account for your business transactions.
- Maintain a telephone listing for your enterprise.
- Maintain time records of your business activities. A diary is sufficient.
- Obtain separate insurance policies for your business activities.

Here's one more cheerful note about your relationship with the IRS: Assume that your activity, be it a hobby or business, is centered on creating a new product. The cost of developing such a new product may be ruled a *capital expenditure* rather than an expense by Internal Revenue. You may have to fill out an extra schedule in April, but it could be worth it for the savings.

In the meantime, keep records that the IRS may wish to see, send for the IRS literature which is designed to help you (the list of booklets and the order forms can be found on the back of the Form 1040 instructions every taxpayer gets in the mail in January), and cross your fingers. *ROM* will try to answer any questions you fire off to us. Keep them coming. ▼

IF YOU'RE MOVING. . .

Please attach here the mailing label with your old address

Write your new address here please.

Name __________

Address __________

City ____ State ____ Zip ____

ROM Publications Corp., Rte. 97, Hampton,
CT 06247.

Building a better computer wasn't easy. But we did it.

Introducing the MSI 6800 Computer System

When we set out to build the new MSI 6800 Computer System, we knew we had our work cut out for us. It had to be at least as good as the now famous MSI FD-8 Floppy Disk Memory System which is also pictured below. So, the first thing we did was analyze all the problems and drawbacks we had encountered with other 6800 systems, and then put our engineers to work on solutions. The objective: Build a better computer.

FD-8 DISK MEMORY
Midwest Scientific Instruments
MSI 6800 COMPUTER SYSTEM
Midwest Scientific Instruments
Olathe, Kansas
Power

We started with power supply. We had big ideas, so we used a hefty 18 amp power supply. You can run full memory and several peripherals without the worry of running out of juice. We also put it in the front of the cabinet so it's out of the way.

The next step was the CPU Board. A separate baud rate generator with strappable clock outputs allows any combination of baud rates up to 9600. A separate strappable system clock is available and allows CPU speeds of up to 2 MHz. The new MSI monitor is MIKBUG software compatible, so you will never have a problem with programs. Additional PROM sockets are available for your own special routines and to expand the monitor. The CPU also contains a single step capability for debugging software.

When we got to the Mother Board, we really made progress. It has 14 slots to give you plenty of room to expand your system to full memory capability, and is compatible with SS-50 bus architecture. Heavy duty bus lines are low impedance, low noise, and provide trouble-free operation.

With all this power and potential, the interface had to be something special. So instead of an interface address in the middle of memory, we put it at the top . . . which gives you a full 56K of continuous memory. Interfaces are strappable so they may be placed at any address. An interface adapter board is compatible with all existing SS-50 circuit boards and interface cards. All MSI interface cards communicate with the rear panel via a short ribbon cable which terminates with a DB-25 connector. All baud rate selection and other strappable options are brought to the connector so they may be automatically selected by whatever plug is inserted into the appropriate interface connector. Straps may also be installed on the circuit board.

To complete the system, we used an MSI 8K Memory Board which employs low power 2102 RAM memory chips and is configured to allow battery back-up power capability. A DIP switch unit allows quick selection of a starting address of the board at any 8K increment of memory.

If you're one of those people who understands the technical stuff, by now you'll agree the MSI 6800 is a better computer. If you're one who does not understand it yet, you'll be more interested in what the system can do . . . play games, conduct research and educational projects, control lab instruments, business applications, or just about anything else you might dream up that a microcomputer can do. The point is . . . the MSI 6800 will do it better.

The MSI 6800 Computer System is available in either kit form or wired and tested. Either way, you get a cabinet, power supply, CPU board, Mother board, Interface board, Memory board, documentation, instructions, schematics, and a programming manual. Everything you need.

There is more to say about the MSI 6800 than space permits. We suggest you send for more information which includes our free catalog of microcomputer products.

Building a better computer was not easy. Becoming the number one seller will be.

See the MSI 6800 Computer System at Personal Computing '77 - Atlantic City.

Midwest Scientific Instruments

220 West Cedar • Olathe, Kansas 66061
913/764-3273 • TWX 910 749 6403 (MSI OLAT)

Midwest Scientific Instruments
220 W. Cedar, Olathe, Kansas 66061

NAME ____________________
ADDRESS ____________________
CITY ____________________
STATE ____________________ ZIP __________

0040877

IT TOOK THE MEDIA TO KEEP THE GOVERNMENT HONEST. BUT IT TAKES "MORE" TO KEEP THE MEDIA HONEST.

MORE watches the media while the media watches you

What if the news reporters, TV commentators, gossip columnists and media gurus who helped write the Watergate story did as thorough a job investigating their own business?

What if Woodward and Bernstein found a Deep Throat somewhere in the bowels of the Times?

Or if Barbara Walters put Barbara Walters under the microscope?

That's the kind of thing that happens each month in the pages of MORE, the Media Magazine.

MORE covers the media like the media itself covers a big story. By looking for sources and listening and digging and watching every word and reading between the lines. By getting behind the scenes and into the back rooms and conference rooms, providing the stories behind the stories you get and the stories behind the stories you never get.

At MORE, we get media people to tell us things they'd never tell anyone else. And we get the very people who report, comment and advertise to write things about reporting, commentating and advertising they could never write anywhere else.

For example we've explored a family feud at the *Times* that may have been responsible for Daniel P. Moynihan becoming a U.S. Senator. We examined the power of a handful of editors at *Time* and *Newsweek* to create and destroy rock stars overnight. We interviewed the controversial *Los Angeles Times* reporter who said that journalists should "lie, cheat, steal or bribe to get their story." And MORE ran the Nora Ephron media column that *Esquire* killed and the profile of Rupert Murdoch that *New York* wouldn't run.

We've given the business to the news business, advertising, movies, publishing and the entire communications industry. And don't think they haven't started watching their words a little more closely now that they know someone else is.

So if you subscribe to the idea that someone should be watching the media like the media watches everyone else, subscribe to MORE.

I WANT TO KEEP THE MEDIA HONEST. SEND ME MorE.

Please enter my subscription immediately.

☐ $12 for 1 year (12 issues)
☐ $21 for 2 years (24 issues)
☐ $30 for 3 years (36 issues)

☐ Payment Enclosed ☐ Bill me

Name
Address
City State Zip
Signature

MorE THE MEDIA MAGAZINE P.O. Box 955 Farmingdale, New York 11735

MR57

Add $2.00 per year for outside U.S. and Canada. Please allow up to 4 to 6 weeks for delivery of first issue.

Memories Are Made of This

by Lee Felsenstein

Why isn't a calculator a computer? Lots of people think it is, and they're partly right. After all, any computer has an arithmetic processing section which duplicates the functions of any calculator. The major difference lies in how the numbers get into and out of the processor.

In the calculator the numbers and commands come directly from the keyboard, as fast as you can hit the keys and read the answers. That's maybe five seconds for each calculation. But the processor gets its work done in a few millionths of a second. The rest of the time is spent waiting for the dumb human to fat-finger another number into the processor.

In the computer the information to be processed and the commands which direct the processing come from an electronic memory which can provide a new number in a millionth of a second or less. That's considerably faster. Not only that, but the instructions themselves can be changed by the computer while it is in operation. Try getting your little hand-held jobbie to do that.

While some calculators may have small memories which can serve as temporary storage for numbers during a calculation, or can swallow and remember a list of commands, a computer is marked by a large, fast memory which can store both numbers and commands and which the computer can write in as well as read.

There's nothing magic about computer memories. They don't handle numbers (which are imaginary), but actually handle patterns which represent numbers. These patterns are made up of elements which have only two possible conditions (on or off, high or low, plus or minus). One such element might be a magnetized spot on a length of recording tape; it can be magnetized with the "north pole" pointing one way or the other. On playback it would then cause a current pulse of positive or negative polarity. The tape would therefore be a memory capable of storing this single "bit" of information as represented by the magnetic spot. And in fact that is exactly how those reels of computer tape store millions of numbers.

Other things besides magnetic impulses on tape have been used for computer memories. UNIVAC 1, built in 1950, used an "acoustic delay line" memory in which pressure impulses traveled down a tube filled with mercury, were picked up by a sort of microphone at the end, then amplified electronically and sent around to the beginning where they were converted back to pressure waves. In this way a pattern of bits could be stored as a recirculating pattern of pressure waves. If you can store a pattern you can store a number.

How large a number can you store? That depends on the number of bits in the pattern. One bit can represent two values. Two bits can cover twice as many values. Three bits can represent eight values. Each time you add a bit you double the number of possibilities. There are limits, of course. Every computer has a maximum "word size" which limits the number of bits in the "word" which is handled inside the machine. A word of sixteen bits can have 65,536 different values—enough for most practical purposes! Actually, when words reach that length it becomes helpful to break them down into more manageable units.

One such unit is the "byte," generally accepted as consisting of eight bits. A byte therefore has 256 possible values, enough for all the letters of the alphabet, in lower case and upper case, all the common punctuation symbols, the numbers 0 through 9, thirty-two special-function "control characters" (like "typewriter-carriage-return"), and lots more. You can see why the byte is a handy unit of information when you are dealing with the printed word.

There are a lot of ways of storing and retrieving the patterns handled by the various types of memories. All of them have their built-in drawbacks. The old standby for storing large amounts of information—magnetic tape, for instance—has to spool through a lot of unwanted information before it gets to the place you want. It's like trying to find your favorite tune on the top 40 cassette—time consuming. This *sequential access memory* of tape compares to an ideal *random access memory* (RAM) where each word is available instantly whenever it's asked for.

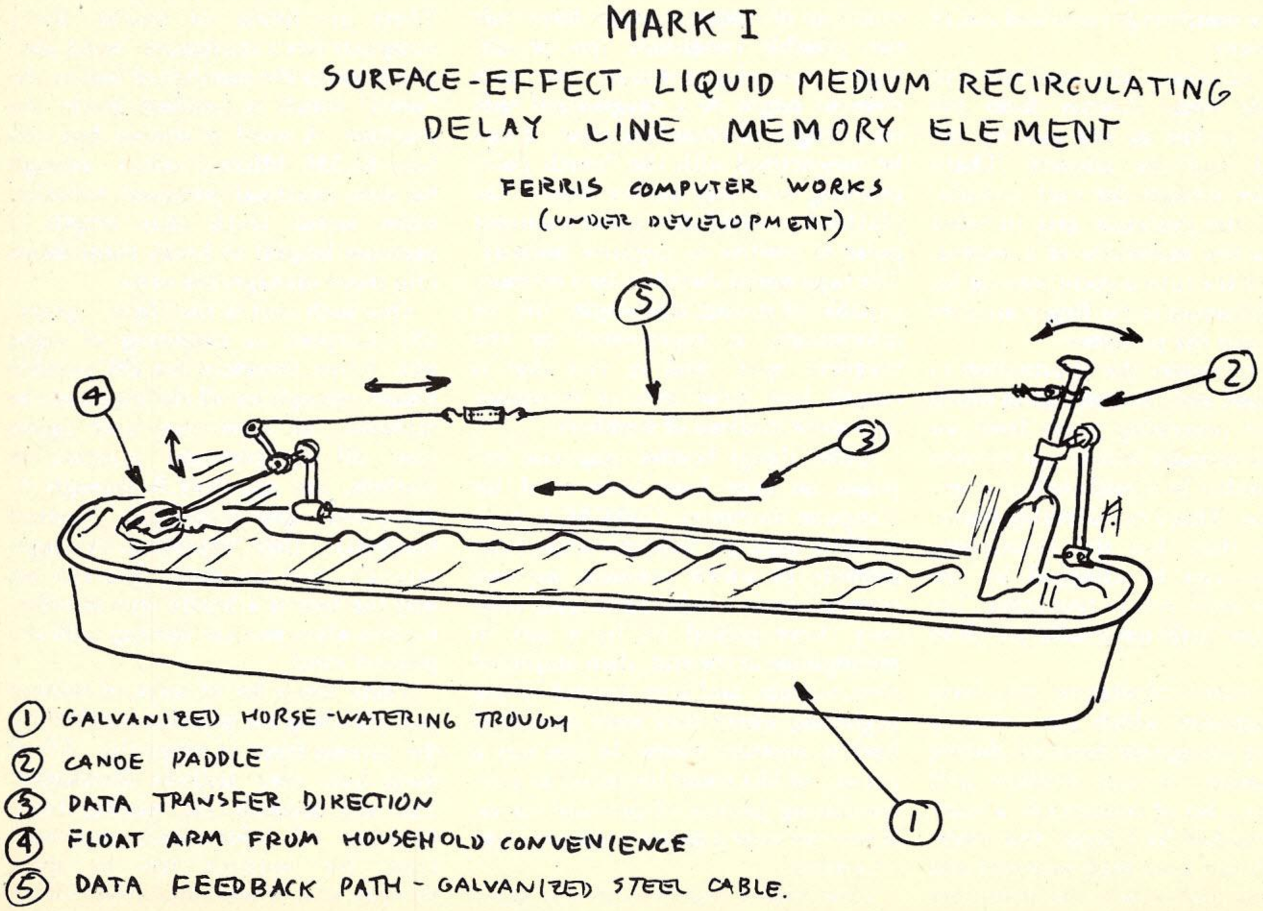
Which brings up an important point. It's not enough to be able to store a pattern; you have to keep track of which pattern is which. This means that each word stored in memory has to have an address, very much like your house address. Memory addresses are therefore numbers represented by patterns (usually electrical) which are presented by the processor and to which the memory responds. If the memory is of a writeable type, the processor can enter a new word to be stored at the address.

Some memories however cannot have information written into them more than once. These are called read-only memories (ROM) and they are used to store unchanging sequences of instructions to the processor, or tables through which the processor will look to find useful information, such as, for instance, multiplication tables. Some ROMs can be erased (with x rays or ultraviolet light) and can be rewritten, usually much more slowly than they can be read. These EPROMs (Eraseable Programmable ROMs) are very useful for developing programs, when it's likely that you'll need to change something but want to keep it around for a while once it's down pat.

ROMs and EPROMs are made with integrated circuit technology, and require that the writeable random access memory be kept supplied with power in order to preserve the patterns. Lose power and you lose memory. Such memory in which the information can disappear (usually when least appropriate) is *volatile memory.* Magnetic storage techniques where the memory units retain their own charge are about the only non-volatile ones available for rapid-writing memory. Almost evey medium has been explored for magnetic storage, including tape, magnetic drums, disks and floppy disks ("diskettes"), and one of the old standards, magnetic cores.

Cores were all the rage when I was a tad. They are tiny doughnuts of a magnetic material (like ferrite) through which are passed three wires. When a pulse of current goes through two of the wires, it magnetizes the core in one direction. Two simultaneous pulses of current in opposite directions cause the direction of magnetization to reverse itself inside the core. This reversal results in a small pulse of current picked up by the third wire,

A lot of old computer folk refer to "core" memory, whether or not the little round beasties are in there.


MARK I
SURFACE-EFFECT LIQUID MEDIUM RECIRCULATING DELAY LINE MEMORY ELEMENT
FERRIS COMPUTER WORKS
(UNDER DEVELOPMENT)
5
4
3
2
1
① GALVANIZED HORSE-WATERING TROUGH
② CANOE PADDLE
③ DATA TRANSFER DIRECTION
④ FLOAT ARM FROM HOUSEHOLD CONVENIENCE
⑤ DATA FEEDBACK PATH - GALVANIZED STEEL CABLE.

Drawings by Lee Felsenstein

FREE-FALL FERRIS, FORMER RESEARCH ASSISTANT IN U.S. SPACE PROGRAM,
(SEE SEPT. 1956 MAD MAGAZINE - DRAWN BY WALLY WOOD)
LEFT A PROMISING CAREER IN CANCER
RESEARCH TO FOUND THE FERRIS
COMPUTER WORKS IN 1962. HABITUALLY
ATTIRED IN LAB COAT AND CLIPBOARD,
HE SPECIALIZES IN MAKING COMPUTER
COMPONENTS AND PERIPHERALS WHICH,
THOUGH CONSTRUCTED FROM READILY-AVAILABLE
MATERIALS, NEVER QUITE WORK.

which is called the "sense" wire, (sensibly enough). If the computer is only reading a word from the memory, it has to rewrite the pattern back into the cores, since the only way it can read is by destroying the contents of an entire word and seeing which bits change. Memory of this sort is *destructive read-out memory.*

Since magnetic core memory is non-volatile, it is useful when the computer must start up with the correct data already in its memory, without waiting for the memory to be filled from, say, a tape or disk. It's expensive stuff to produce, though, what with all the needle-and-thread stitching they have to do to string wires through thousands of cores (there's one core for each and every bit), so it hasn't made it to the low-cost personal computer market yet. Still there are a lot of old computer folk who automatically refer to the high-speed memory of any computer as "core memory" whether or not the little round beasties are in there.

Magnetic core manufacturers are scheming to find ways to get their little doughnuts into your hands.

Some of the first cheap integrated circuit memories were shift registers, which are sequential devices structured internally like bucket brigades. The buckets could have a bit in them or not, and the pattern was passed around and around. These early shift registers were volatile, so called *dynamic* shift registers. If the shifting stopped, the information vanished. So the memories were designed to keep whirling the patterns around forever until the processor requested them. The processor would, of course, have to wait until the right bits came around before it could pick them up and go about its business.

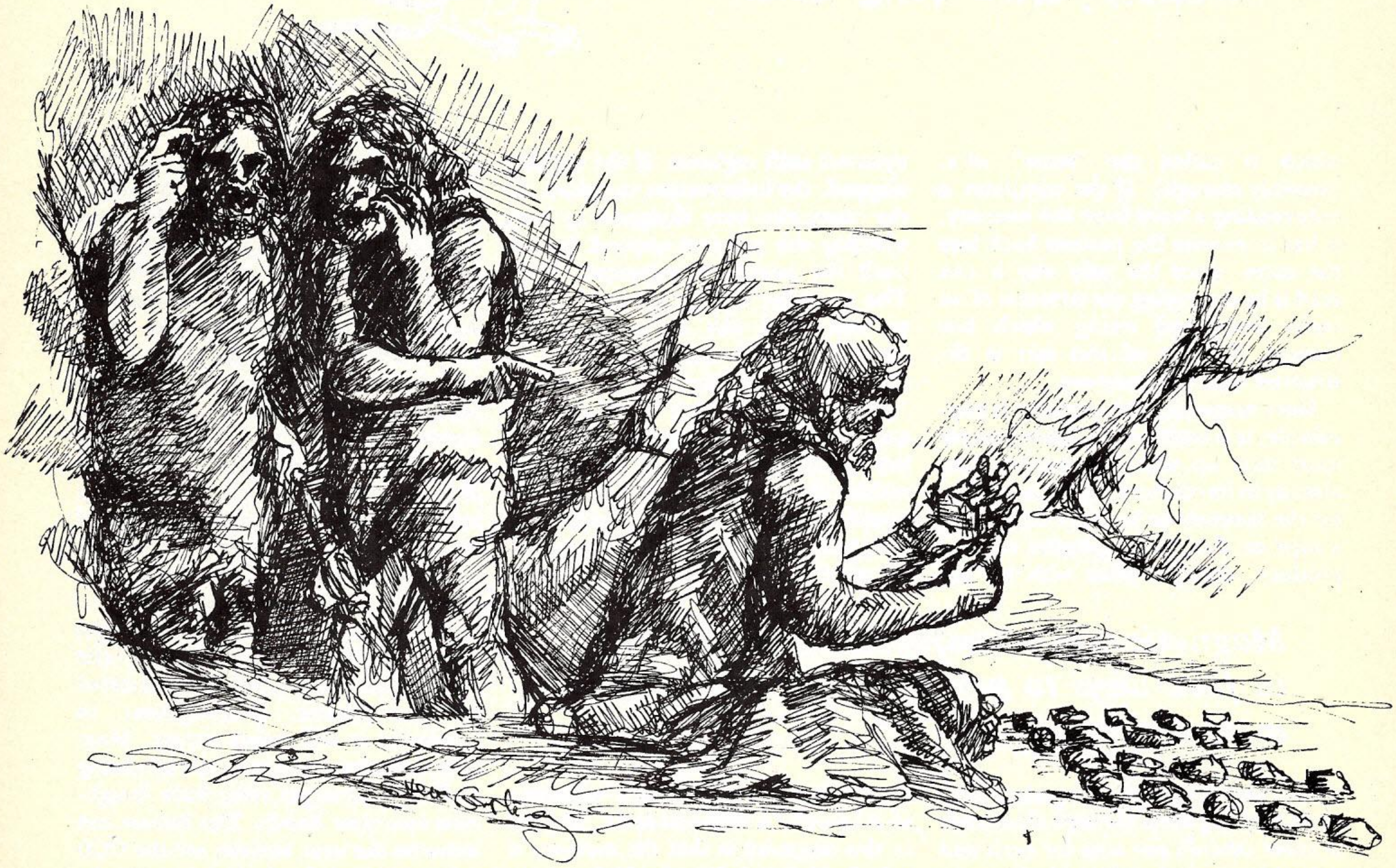
After shift registers came the dynamic RAMs which required much less waiting time. In this technology, rather than wait for the merry-go-round to come around again, the processor just asks for its information by address, and gets it after a very short delay (usually about a millionth of a second, a *microsecond*). The fly in this ointment is that the memory is volatile if it is left to itself for more than a few thousandths of a second. If it is asked for a sequence of addresses periodically, it must remind itself what all of its stored words are (this is *refreshing* the memory). This adds a complication to the design of the memory and means that idling conditions for it are somewhat more complex than just a cessation of all activity.

Then, shortly after the advent of dynamic RAM, there arrived the *static* RAMs. These integrated circuits had more inside them, but didn't need the refreshing which dynamics needed. They are good memories for amateurs to work with. Static RAMs need nothing but power (and not even their full operating amount of power) to retain their patterns. The drawback of static RAMs is that they take more power per bit and usually hold about one fourth the contents of dynamic RAMs. The cost per bit of dynamic memory is less than that of statics as well. But the complexities of the refreshing process (which can get very subtle indeed) had kept them out of the personal computers until just recently, when the first 16K byte memory boards were announced.

But don't think that you can get away with forgetting everything else this article said, now that we've arrived at the present. Development in memories is still going strong. Magnetic core manufacturers are scheming to find ways to get their little doughnuts into your hands. The hottest two items on the near horizon are the CCD (charge-coupled device) and the magnetic bubble memories, both of which are of the sequential access type, and represent new and better ways to make a shift register. We're getting back on the merry-go-round. Of course the CCD is huge and the magnetic bubble is non-volatile, but many old, forgotten hardware and software techniques will be revived or reinvented to compensate for the drawbacks of these shiny new devices.

Any bets that we'll be playing with mercury-column acoustic delay line memories in a few years? ▼

Memory, Memory,



the invention of the digital computer
by Ugrok, 12,036 B.C.

How Much Memory?

by Stan Veit

Our cave-man ancestors counted on their fingers and toes. They extended their memory with pebbles and sticks and you might say that this developed into the first computer—the abacus. The earliest mechanical calculators used gear teeth for memory. (Need more memory? Just add more teeth or a larger gear.) The first digital computers used vacuum tubes. Even the small amount of memory they had occupied racks and racks of electronic gear. Now we have microcomputers that use relatively cheap, abundant semiconductor memory chips and all we have to do to extend our memory is to plug in an extra memory board.

Given the technological glut of available memory boards, and money to pay the bill, we then may ask, "How much memory do I need in my computer?"

The answer to this question is quite simple, "You need more than you have."

In fact, no one in the history of computing has ever had enough memory. The reason is obvious: as soon as you expand your memory capacity, you think of nifty new programs in order to use it. This gets you involved in new ideas, and before long, lo, you are memory-bound once more!

For a long time we thought we knew the limits to our memory capacity. Most of our computers are eight-bit machines with a sixteen-bit address bus. Therefore there are 2^{16} or 65,536 possible memory locations that can be uniquely addressed in a binary system. We usually say that can have up to 64K of memory, just as we say, "It is so many light years to the nearest star," (an amount so large we never hope to reach it!).

The first personal computers were equipped with 1K of memory, or even 2K of memory and the hobbyists were able to write machine language programs to play simple games and dream of running BASIC.

Soon 4K memory boards came along with dynamic chips that used lots of power and sometimes ran stealing cycles from the microprocessor and dropping bits all over the place. You could spend an evening toggling in a program from the front panel switches and if it ran, it rarely did what it was designed to do. At last, the wizards of Silicon Valley came up with 1K-by-one-bit static RAM chips, and the easy-to-build 4K RAM board was available to plug into everyone's computer, at a reasonable price too!

The supply of memory boards led to the development of more software and soon Tiny BASIC appeared, then 4K BASIC and the many versions of 8K BASIC. The hobbyists now had an easy-to-learn language that almost anyone could program, and they proceded to write all kinds of games and so-called "practical programs."

In no time at all the 4K RAM boards in the computer had reached the limit of the electrical power in the computer, and then the available slots were filled. The hobbyists kept asking for more board space. The first personal computers came with four- or six-slot mother boards, just enough for the MPU board, front panel board, a memory board, and an I/O board to talk to the outside world. The manufacturers learned that if they were to sell the new memory boards and all the other new modules they had developed, there must be some place into which to plug the boards. Thus the twenty-two-slot and eighteen-slot mother board became standard, and the first thing the hobbyist added was more memory.

When the user had only a little memory, he wrote clever little machine language programs, priding himself on his ability to do something efficiently and elegantly, using just a few lines. Now there was BASIC! The average hobbyist started to write much more elaborate routines. The first Tiny BASIC lacked many features that the ambitious BASIC programmer felt he had to have, and the software and hardware manufacturers responded with all the "bells and whistles," used on the big machines. Strings, matrices, nested variables, floating point, double precision math, PRINT USING, and other functions and commands

Drawings by Steve Gerling

were added to increase the power and data-handling ability of the microcomputers. BASIC grew with the addition of all these features.

When you are using machine code, the program occupies a minimum of memory, and the processor is able to act directly upon the instructions you place in each memory location. If you are using a high-level language, the computer memory must support (contain) the BASIC interpreter. In addition the memory must support the application program written in BASIC and, finally, the data entered by the user. In this case, note that the use of BASIC (or any other high-level language) has required three times as much memory space to do the same task as a program written directly in machine language. Not only does it require more memory, but with each level of language, there are "housekeeping" routines (such as Loaders) that also take memory space.

No one in the history of computing has ever had enough memory.

Thus, the addition of memory leads to the use of more complex programs, which leads in turn to the demand for more memory. Eventually, this cycle ends at the full amount of memory that the computer is capable of addressing, and then the memory demanded must go to mass storage devices like disks and tapes.

The semiconductor IC manufacturers soon learned how to make chips with greater capacity, first making 4K memory chips and then 16K memory chips. Soon the standard S-100 bus memory board could hold 16K, then 32K, and even 64K of memory. At first the prices of these big memory boards kept them out of the hobby market, but their development had another indirect effect. The 1K and 4K chips became much cheaper and the market became full of 8K and 16K memory boards at very low prices. Now there are fully built and tested 8K boards on the market for $229, a drop of over a hundred dollars from the prices of only a few months ago. The price of 16K boards fully-built and tested will soon break the four hundred dollar level. In addition, soon the large semiconductor manufacturers will start to market S-100 bus memory boards under their own names. Then the usual pattern of discounting will occur as the large companies move into the market.

The question is, who is going to use all these memory boards? The answer to that is found in the expansion of the market through the entrance of new suppliers and retailers at the chain-store and department-store level. In addition, magazine and book publishers are featuring free software which has business as well as home applications. As all this new technology is used, the demand for more memory will grow and grow.

Since an eight-bit microcomputer with sixteen address lines can only use about 64K of memory, how can the demand for more memory be met? Well, the designers have come up with a new scheme for organizing the memory of the computer by dividing it into blocks and addressing a whole block of memory from one address location. In this way the limits of memory capacity are fixed only by power consumption and cost. The advertising slogan "Megabytes of Memory" has already become familiar to the readers of computer publications. They now can have at their command more computing power than some of the large-scale computers of the recent past!

All of this has confused the hobbyist. His investment in a computer system originally started out at about six or seven hundred dollars and grew to about a thousand dollars as he added more boards or peripherals. Now he is told that for all of the really useful applications he needs more and more memory. Where should it all end? How much memory does the user really need?

Let us analyze the requirements to do useful computing and see if we can set some limits to memory requirements and cost. If we use a page of alphanumeric characters as a unit of measurement, we can get some idea of memory requirements for useful work. Let us consider our page as being the usual letter size, 8½ x 11 inches, or eighty characters wide by sixty-four lines long. (These dimensions are obtained from the type sizes of the

ROMtutorial ROMtutorial

K: 1024, not 1000 as in metric measurement.

Dynamic chips: Integrated circuits ('chips') which must be clocked or pulsed continuously in order to operate properly. Dynamic memory is memory which must be continuously *refreshed,* i.e. rewritten, to hold its information.

Static RAM chips: Integrated circuits used for storing information which use electronic "latch" circuits internally. Unlike the leaky dynamic chips, no refreshing is required for static chips.

Four-slot or six-slot mother boards: Printed circuit boards made for the purpose of connecting other printed circuit boards together. The mother board has several connectors (slots) which can accept the other boards edge-on, so that they stand up at right angles to the mother board. Mother boards are used widely in computers to replace wiring and provide a mechanical foundation.

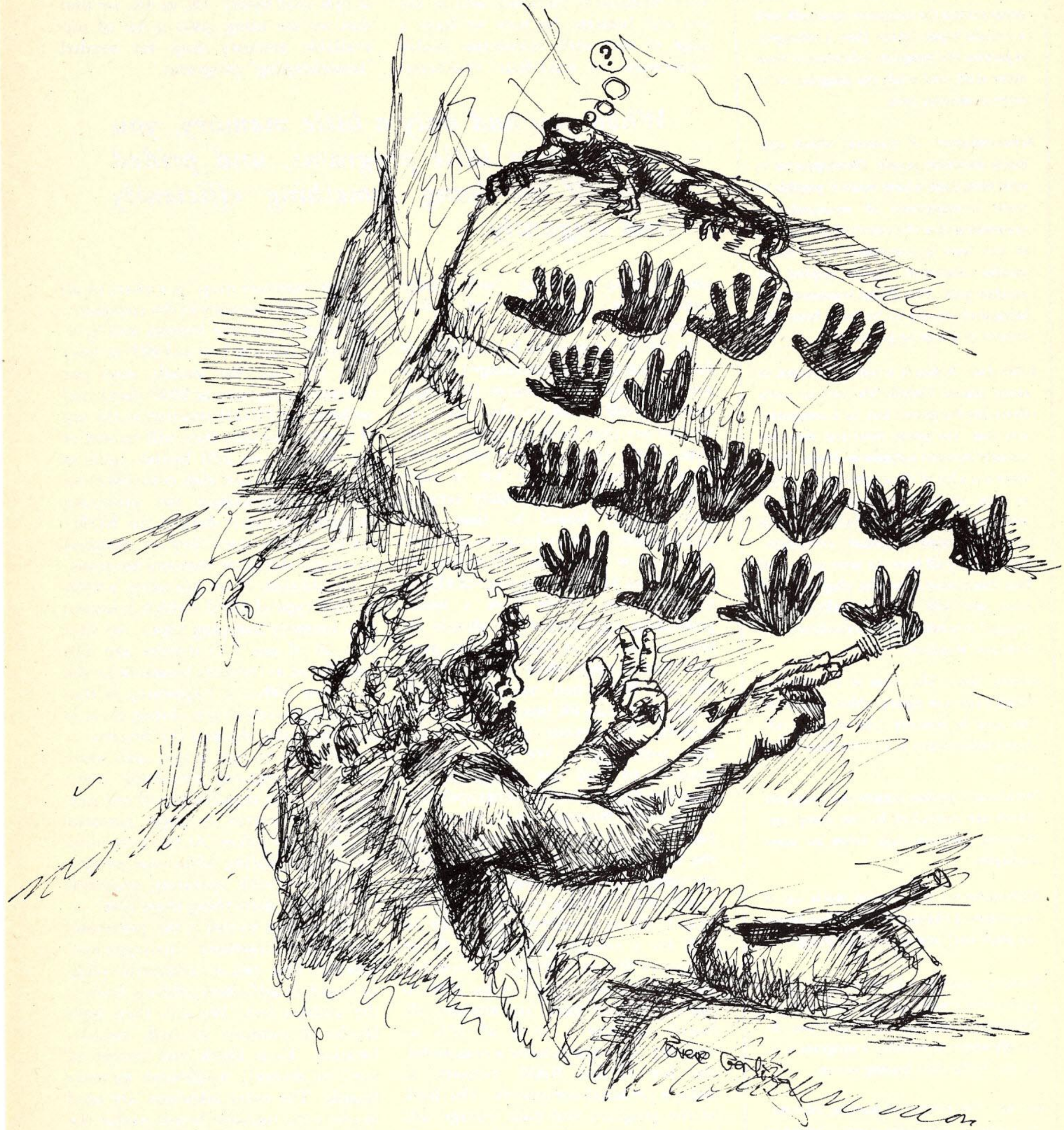
MPU board: The printed circuit board in a microcomputer system which holds the MicroProcessing Unit, which is also called the Central Processing Unit, or CPU. Whatever the name, this is the "brain" of the computer.

Front panel board: The printed circuit board in a computer which contains the switches and lights allowing someone to tinker with the information and programs inside. Not all computers have front panels, which are useful only when the machine is not running.

Strings, matrices, etc.: Powerful features of modern programming languages. Strings and matrices are data structures. A data structure is an organizing principle used to impose order on a collection of data or information. A *string* is a data structure which groups a number of characters into a sequence. A *matrix* is a data structure which groups a number of characters or integers or more complex numbers into a rectangular array like a table.

Nested variables are variables (like algebraic 'x's and 'y's) which appear in complex algebraic expressions containing nested parentheses. For example in the expression: $(a+[b/(1+n/a)])$, "a," "b," and "n" are all nested variables.

Floating point is a representation for numbers which makes it possible to write very large numbers as well as very small ones in a relatively compact way.



11,903 BC. - Ugrok's grandson, Borak,
invents memory board for digital computer...
calls it "art"

ROMtutorial ROMtutorial

Loaders: Programs used to load other programs into a computer. The loader program sets the computer up to talk with a certain input device (like a teletype), separates the program information from other stuff, and sends the program to its correct memory area.

Semiconductor: A material which conducts electricity poorly. Developments in solid state physics have made it possible to make arrangements of semiconductor materials so that the electrical conduction of one part is controlled by a much smaller electrical signal applied to another part. This allows transistors and integrated circuits to be built from tiny chunks of silicon or germanium.

S-100 bus: A bus is a wire connected to many places. Usually it's used to carry electricity for power, but, in a computer, very fast, low-power electrical impulses are sent between sections on buses. Since there are a lot of these signals happening at once, computer buses have a lot of wires in them. The S-100 is a 100-wire bus used by many personal computers. Because they all have the same pattern of four interconnections, the plug-in board from one will work (usually) when plugged into another S-100 machine. The S means "standard."

Address lines: The wires in a computer which carry the signals which determine the area in memory (the memory address) information is to come from or be sent to.

Peripherals: Devices outside the computer which are controlled by the computer. Printers, teletypes, tape drives are some examples.

Alphanumeric: Information made up of sequences of characters containing letters (alphabetics) and digits (numerics).

Bootstrap loaders: Are used to get programs into a machine for a first start-up. Relocatable loaders enable the user to specify where in memory a program is to be put at the time loading occurs.

Sorting: The process of putting information into a particular order. Numeric sorting orders information by its numeric value, in ascending (increasingly greater) or descending (increasingly smaller) value. Alphabetic sorting puts information in alphabetic order.

familiar office typewriter, and are just for a basis of comparison.) When we compose a page of text, we always leave margins at each side and at the top and bottom. So now we have a page of text with maximum usable dimensions of sixty-four characters wide by sixty lines long. Such a page contains a maximum of 3,840 characters.

> *When you had only a little memory, you wrote clever little programs, and prided yourself on doing something efficiently and elegantly.*

In ASCII (American Standard Code for Information Interchange), there are eight bits per character (although seven bits will define the character), so we can say that each character is one byte (eight bits). In our microcomputer we can put one bit in each memory cell and we usually arrange our memory board so that each addressable memory location contains one byte or eight bits. Therefore we need about 4K to hold a page of alphanumeric text. If we have a word-processing application, we should be able to refer to at least the last page and current page while we are editing the text. Now, then, our current data needs at least an 8K board, (or two 4K boards). Remember, this is only our data work space. We need enough memory to load our programs (remember Loaders and Bootstraps) and the BASIC program itself, say about 8K. Then we need memory space for the text-editor program, say 6K, and enough space to manipulate the data, so now we're talking about another 8K board. All of a sudden we are up to 24K for a simple application. If we expect to do any sorting, we need additional memory space to be able to move blocks of data, say another 8K board, making a total of 32K of memory. This seems to be a reasonable size for on-line RAM memory in today's personal computers. The bulk of the program and data storage will be in off-line mass storage devices such as tapes and disks.

Which brings us to another consideration. As we add such things as tape and disk interfaces and controllers, we add read-only-memory (ROM) to control such devices. Most of this is in the form of programmable-read-only-memory (PROM or EPROM). Since these chips are made in standard sizes of ½K (512 bytes), 1K, or 4K, we find that we are using quite a bit of our available memory map for needed "housekeeping" programs.

The "memory map" is a chart of all the available memory in the computer. Location 0 is at the bottom and location 64K (referred to as FFFF in hex) is at the top. We usually start our operating programs at 0000 (hex) and we keep our PROM routines at the top of memory so that they will be out of the way. As we add boards such as video modules and disk controllers we find that sometimes the operating programs on such devices (in ROM) are in conflict, since they are designed to occupy the same memory locations.

In addition, if we are using a 6800 or 6502 type of MPU (which is known as a memory-mapping type), we find that all of our I/O devices are also considered as memory locations by the processor. What is happening is that our memory capacity is closing in on us from both the top and the bottom.

We want to run longer and more complex programs requiring extra memory. The manufacturers are providing the boards in larger amounts and at lower prices. At the same time we are also adding additional peripheral devices with operating programs on PROMs. Something must give.

The answer to this is the previously mentioned "memory management" scheme. This uses an additional plug-in board to add extra address lines to the address bus. We can then have blocks of memory at each memory location. Each block can consist of one, or several, additional memory boards. The extra addresses are used to select the memory board within the block and the addresses on the board. In some cases, this additional is housed in a separate chassis with its own power supply. That relieves the power drain on supplies never originally designed for so much memory.

The demand for all this memory has

also resulted in the return to popularity of the dynamic type of memory chip. The first large-scale memory chips were dynamic since they were easier to manufacture. A dynamic memory chip only retains its charge (information) for a very short time. Therefore it must be recharged (refreshed) at regular intervals. This function was originally assigned to the microprocessor which had to stop what it was doing and refresh the memory. This made for timing problems and lost bits and it also slowed down the computer.

When static chips were developed, everyone abandoned the dynamic type. The static type of memory retains its charge (information) only as long as the power is not turned off, or as long as it is charged by the computer under program control. Static chips also use more power. As larger-capacity memory chips came into use, the power supplies of the older microcomputers became overloaded mainly because of the increased demand of static memory.

Today's 4K and 16K memory chips are mainly dynamic or semi-dynamic, requiring some refresh. However, the memory boards are now designed with the refresh circuits on the board itself so they do not interfere with the operation of the MPU. In addition, the newer microprocessor chips such as the Z80 have provision for refresh without interfering with the operation of the MPU. Thus the second barrier to large memory expansion, the requirement for refresh from the processor, has been minimized.

The addition of memory leads to more complex programs, which leads in turn to the demand for more memory.

The third limitation to memory expansion has been the cost. For the hobbyist or small business man, there are so many goodies competing for his dollar that he doesn't know what to buy first. He does know that he needs more memory, but what about all those costs? Well, in the good old days (last year), he added memory in 4K increments at $139 per board (more or less). Now the manufacturers are selling 8K, 16K, 32K, and 64K boards. The price-per-K is way down, but the total he must add at one time is way up. Also the hobbyist wants to get all the neat new gadgets and more memory when he can, so he must preserve his slots in the mother board for the future when he has more money. If he adds memory in 4K increments he will run out of space very fast. But if he buys 16K or 32K boards he may save slots and then run out of money! Perhaps the answer to this is to buy the 8K boards which are very cheap right now, and then later use a memory-management scheme to expand the memory of the entire computer.

The other solution is to put any expansion money he has into mass storage devices instead of RAM. The prices of floppy disks and mini-floppies are coming down very fast and the storage capacity is going up. The manufacturers of disks are now introducing double-density, quad-density, and disks which play on two sides. In a short time the mini-floppy will hold more than today's standard floppy, and the larger floppy will rival the disk pack! Earlier we suggested that 32K was a good size for a RAM memory in some cases, and this holds true for systems with fast-access off-computer mass storage capacity.

Besides, within a year or two, the floppy disk and the cassette may give way to non-mechanical devices such as the bubble memory, and then our external memory may expand beyond our present dreams. But we will still need RAM within the computer to execute all the programs we can store in the mass storage device. We still can expect the prices to go down and the need and wants to go up until megabytes are actually the accepted normal memory capacity for small machines.

The next step will follow the trend set by the big computers. If you keep adding addressing lines, you can keep adding memory locations. As the techniques for manufacturing new and larger chips improve, our microcomputers will change from eight-bit to sixteen-bit and then to thirty-two-bit machines. Tomorrow's "Table Top 370" may well use a megabyte of 32K-bit memory! Then some one will come out with an expansion module! ▼

ROMtutorial ROMtutorial

Read Only Memory: memory like a telephone directory which can only be read by the computer and not written in. It's used to hold instructions for the computer (the program).

Erasable programmable read only memory: A chip which will hold information indefinitely after being written, but which will forget the information if exposed to ultraviolet light. (The cover of the chip is clear quartz.)

FFFF(in hex): Hex is short for Hexadecimal, which is how we would count if we had sixteen fingers. FFFF is the largest possible hexadecimal number composed of four digits. Its actual value is determined as follows:

FFFF = (15x4096) + (15x256) + (15x16)
+ (15x1) = 65,535

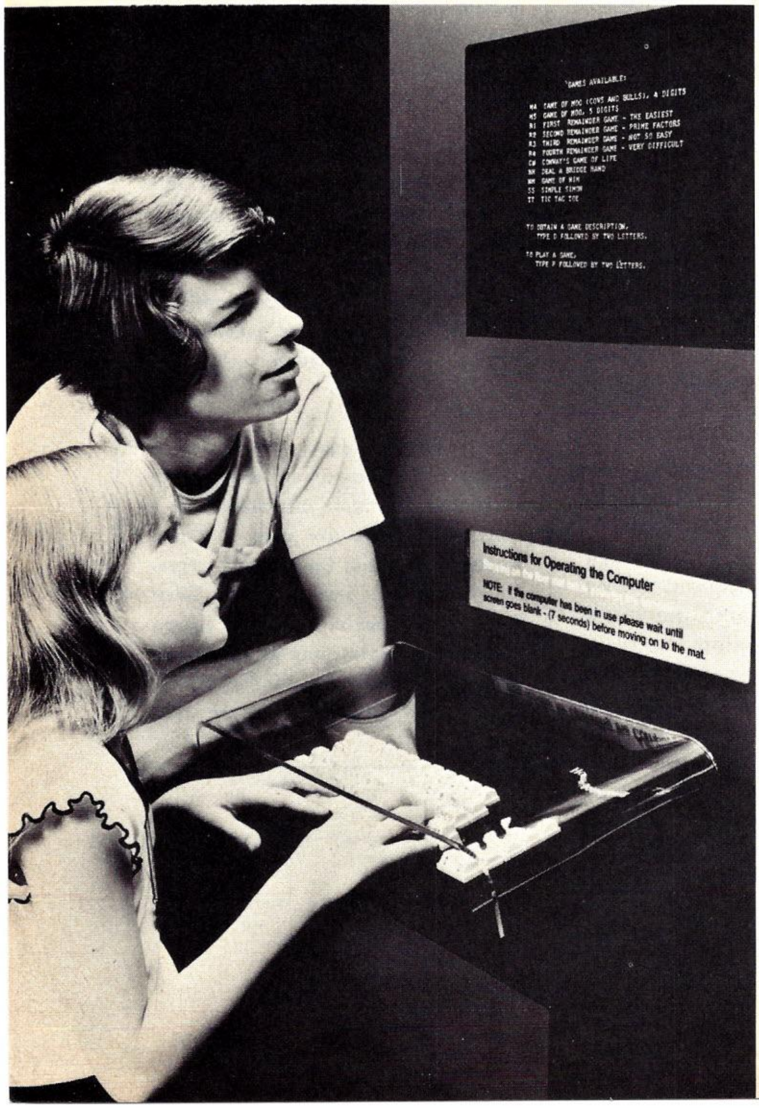
Disk controller: Electronic hardware which allows the computer to control a disk memory device. The controller must keep track of the rotary position of the disk platter, the track position of the moving head, and must arrange the data received from the computer into a serial form which can be recorded on the disk and vice versa.

Floppy disk: An information storage device which uses a "disk" which looks like a 45 rpm disk but is coated with a magnetic surface like that used on a recording tape. The disk is flexible, or "floppy." The disk drive rotates the disk and can position a movable head which can magnetically write and read information on the disk.

Double density, quad density: The attempt is always continuing to be able to store more and more information on a magnetic medium like disks or tapes. Double density and Quad density represents twice and four times the usual density of "bits per inch" at which a given disk or tape system may be capable of storing information. What is double density this year will probably be the usual next year.

Alice through the Video Terminal or An Electronic-Metaphysical Fantasy

by Sally Steinberg



```
'GAMES AVAILABLE:

M4  GAME OF MOO (COWS AND BULLS), 4 DIGITS
M5  GAME OF MOO, 5 DIGITS
R1  FIRST  REMAINDER GAME - THE EASIEST
R2  SECOND REMAINDER GAME - PRIME FACTORS
R3  THIRD  REMAINDER GAME - NOT SO EASY
R4  FOURTH REMAINDER GAME - VERY DIFFICULT
CW  CONWAY'S GAME OF LIFE
BH  DEAL A BRIDGE HAND
NM  GAME OF NIM
SS  SIMPLE SIMON
TT  TIC TAC TOE

TO OBTAIN A GAME DESCRIPTION,
   TYPE D FOLLOWED BY TWO LETTERS.

TO PLAY A GAME,
   TYPE P FOLLOWED BY TWO LETTERS.
```

Instructions for Operating the Computer
NOTE: If the computer has been in use please wait until screen goes blank - (7 seconds) before moving on to the mat.

What if Alice-in-Wonderland had fallen through a terminal instead of a rabbit hole or a looking glass? Instead of the pack of hearts-spades-clubs, she would have found punch cards playing Fortran, not croquet; she would have worked on planning Mushroom Program II for Pituitary Increase, and we would know about Disembodied Feline instead of the Cheshire Smile. A modern dream-come-true, dreamed not on a grassy bank, but on a bed programmed to eject the sleeper at the correct time.

Photographs courtesy of Museum of Science, Boston

This fantasy is the result of a week I just spent in search of a computer I could understand. Some time in midweek, a screw came loose—perhaps the inevitable result when ignoramus confronts computer. Knowing nothing about computers to begin with, I set myself the task of learning something of what they were all about in only one week. Spurred by the derogatory comments of my nine-year-old son ("Don't you even know a computer when you see one, Mom?") and other insults, I set about, through a series of adventures and misadventures, finding terminals and chips and what would happen if I "cleared." I decided to see what I could learn about our future dictators, these versatile boxes.

I also had a bad reputation to reverse. Numbers are not my thing, and my son thinks I am a mathematical idiot. I am famous for my checkbook calculations—five hundred dollars from three hundred dollars equals five hundred dollars, plus overdraft and angry husband.

An unsuspecting soul, I found myself face to face with the unfathomable machine-with-a-memory, an unknown and unknowable mystery. Later a computer salesman described the computer to me as the only machine man has invented to help his brain. Besides bedazzlement, my week of seeing what I could learn about computers brought me many splendors.

Alice with punch cards playing Fortran, not croquet?

Where do you learn about an esoteric scientific subject like computers? I heard by the grapevine that there is a small computer in the Boston Museum of Science. My telephone call was graciously received. A museum person led me to a series of typewriter-sized boxes. I submitted one to a series of elementary tasks to prove I could add and subtract. After three hours of poring over instructions and carrying them out, I was pleased with myself for figuring out how many words my last article had had without having to add up each page individually.

Triumphant, I returned to my husband, who raised his eyebrow and suggested that I had played with the wrong machine. I don't know a computer when I see one, right? I argued indignantly, but he convinced me I had, in fact, attacked the Wang Calculator, not a full-fledged computer. I went to bed in a huge depression.

Next day I arose to return to the Museum of Science; it was still mobbed with school buses. Brave new world. The only real computer I could find filled a whole room, and played tic-tac-toe with children rather badly. Since this was the only computer the museum personnel could find as well, I accepted this as, perforce, my "computer experience." Heaven was never like this, I thought, as I faced its intricacies.

It seemed that this small miracle could do anything.

Thus began a voyage in search of computers, taking me eventually to the Boston Children's Museum as well, where my family took turns playing with the computer while I recorded my impressions, and my husband looked after our two-year-old.

Ha. Instead, my husband sat mesmerized by the computer game of *Inchworm,* where a set of commas and exclamation points moves up, down, and sideways on a TV screen. I ran after the two-year-old. Only when the computer called him by name and told him, "You have played with this terminal enough, David," was he induced to leave.

I concluded that the power of the machine was greater than mine, and that maybe we needed a computer at home to order him to stop reading the newspaper and come down for dinner.

My erroneous encounter with the Wang Calculator had some benefits. It allowed me to feel that now I had graduated from the elementary to the complex, the calculator to the computer. At the same time I realized that these two machines were different orders of being. It clarified for me what a computer did, as opposed to what a calculator did, even though I knew they overlapped at times. Had I not made that infuriating error, I might only have had a hazy notion of how calculators and computers differ, especially as the Wang calculators are sometimes called computers, even by museum personnel.

My original sense of power at mastering the Wang Calculator had been exhilarating. I had actually sat down at a machine whose sole language was numbers and I had figured it out (stopping, of course, at logarithms, which I gave up for Lent in the eleventh grade and have not indulged in since). It seemed that this small miracle could do anything—I was still green—bragging on its instructions that it took minutes to do electronically what it takes a person hours to do. I am inclined to guffaw at such statements, as at machines in general. What if it makes a mistake?

But I sat at the wooden desk and checked the words in my article against the machine total and, lo and behold, it worked. I even did a few square roots hung over from high school math, just to see, you know, if the machine was as smart as it said it was. So far so good. I concluded from the children swarming around me that this magic number machine, all flashing lights, also had sex appeal and promise for homework improvement, to say nothing of income taxes and checkbooks. I also remembered my husband's face in front of the checkbook—cheerful with his pocket calculator, stormy without.

This electronic slide rule, as the Wang Calculator calls itself, even has a button to fix its own jammed keyboard. By now deft at the keys, I whipped up the cost of milk per ounce (.0278125 at the corner store), and how much I was paying my babysitter per hour to allow me to play with these machines. Unbeknownst to me at the time, the dangerous process of machine addiction was setting in.

The Honeywell Computer at the Science Museum came next. I entered a dimly lit room with insert displays on the walls. Although my natural tendencies led me first to the show of computer components for their aesthetic beauty, I resisted temptation, reviewing systematically the information provided in the other boxes as well.

I learned that there are five basic computer elements—for control, input, processing, storing, and output—and with these you can process any kind of information. Then the machine remembers it and feeds it back in instant solutions to problems.

What problems? That was in another box, showing how medical profiles can help cure chemical imbalances, or how stored weather information can aid in forecasts. How exactly? Another box showed me the number of steps involved in a simple problem—like making twenty-seven cents out of nickels, dimes, and pennies in the smallest number of coins. As I read about how the machine stores coin values, eliminates, accepts, and compares, it occurs to me that it might be simpler to solve it in your head. But when I look at how payrolls are computed, the number of steps looks positively forbidding, with 230 pages of coding, and I decide this problem is best left to the computer, which can probably do a fine job.

In another wall box I see a history of tools for information-processing. This helps me to see where the computer fits into our lives along with older tools like the abacus, cave painting, early writing, and the printing press.

It also helps me to place the computer in the modern consciousness when I hear that the computer is the most popular of the exhibits at the museum. I see that I am the exception, and that lots of children and adults already know about these machines and want to know more. Since 1973 it has been so popular that they say one group of teenagers learned programming by themselves on it. Afterward, Honeywell sent a teacher to teach programming to enthusiastic museum goers.

All the while I am conscious of the children in the background clicking at tic-tac-toe, undoubtedly a great drawing point. I am intrigued with the lure of this game played by the machine—the children stand fascinated for a long time. Me against the computer. It taps our competitive instincts. Children and adults who like toys like computers. *Homo ludens,* we all like to play. Games moving without effort are captivating.

I march doggedly on from the question of what the computer can do to something I seem to have overlooked up to now, what the computer is. I should deal with this in my learning process if I am to face the force of the impact on the world of this machine. I feel that the computer is all-knowing and all-powerful. I know that it shows how to sort out the basic mental steps in a problem. I see that it has an ingenious language where numbers are converted to two-symbol notation on tape for storing, even though the tape just looks like a jumble of holes to me.

This magic number machine, all flashing lights, also had sex appeal.

But before I wound my way this far through the convolutions of discovery I had not really confronted what this behemoth is. The wall display tells me now. It is a "combination of electronic circuits" used to process information and solve problems. But of course.

My adventures in computerland are educating me about these brainy machines, and they are opening my eyes to the power of the computer over what we generally regard as our thinking, feeling fellow creatures, large and small. We live in an age when sometimes more satisfaction comes from talking to a machine than from talking to people. The allure of a machine which uses your name more than people do in this de-personalized age is clear.

We also live in an age where the tool of magic has changed. Now instantaneous change is produced by computers instead of magic wands. The computer is as impressive as the fairy godmother used to be. It is a part of our folklore and of our children's imaginations, a hobgoblin and hero of modern children's fantasies.

"Make me a malted," went the joke in our day.

"Poof! You're a malted," we replied.

Now the computer poofs any program you like, waters plants, plays chess, welcomes you home. The only problem is that someone else makes the programs if you don't know how. What if the progammer's life style differs from mine?

In trying to find out what the computer could do, I was always looking

"All right, let's see now. If you kick in $16 for the thruster and we get RAMCO to cut a sixteenth off the shielding. . . ."

ahead, just as the computer industry looks ahead, to what the machine might do if allowed. Free me for more creative work? Why should I not allow a machine to care for household and other drudgery? We might even eliminate personality clashes and alcoholic domestics.

I was definitely "finding out," getting an unexpected education. Maybe I used to be a Victorian lady writing in my chambers; I had graduated, I was now "with it." What Victorian lady visits computer warehouses?

A wonderful, suggestive analog to humans, these computers. No distractions, just concentration on goals. It makes one think of fantastic possibilities. Push a button, solve a problem. It seems so easy. From there to the idea of carrying an internal computer with a CLEAR button to iron out the bugs in our own psychic systems seems not unthinkable. How efficient we should be if we eliminated human static, the pressure at the office, the aggravations at home. "If only a man," to paraphrase *My Fair Lady,* "was more like a machine." Push CLEAR and program a trouble-free day. Freed of all bugs, perhaps we should all meditate forever and attain eternal peace.

"There's gotta be a way we can get this thing to change that F in physics."

At the Children's Museum in Boston I found a varied set of programs and a great deal of information about the computer. Their computer is a Digital PDP 11/40, and its working parts are on view on the wall, not enclosed behind smoky glass as at the Science Museum.

William Mayhew is in charge, and he articulates the purpose of the exhibit—making computers understandable in comfortable, not fear-inducing, surroundings. Well, that's why I'm here. Children love being able to try the computers, but as for dispelling fear, the impact on adults is greater, states Mayhew. Adults have had more time to associate computers with the chilling worlds of depersonalization and electronic control, and to come to fear them. Children accept computers as part of their world. The adults find the computers here more accessible than they anticipate, and the process of domestication begins.

Freed of all bugs, perhaps we should all meditate forever and attain eternal peace.

The Children's Museum exhibit is a wild success, with ten terminals for public use. The computer processes information for the museum and for ticket services and other cultural organizations in Boston. The most frequent visitor comment is "Why don't you have more computers?"

There are occasional computer workshops for kids, and they can also learn programming indirectly through the games. As with the Honeywell computer at the Museum of Science, this one was also donated. It came to the museum by the good offices of the Digital Equipment Corporation along with the Hayden Foundation and the Bell Telephone Laboratories.

Kids are drawn to the instant result, the push-button magic.

Children have an uncanny way of getting to the heart of a problem without distractions. On the wall at the Children's Museum many questions about computers are asked by children and answered for them. Why are computers made? Because they are "fast, accurate," and they "remember lots of things." Computer language is a "special way for a person to tell a computer what to do." A computer can add two numbers in one microsecond, store large amounts of information in its memory, as many as ten million characters, and understand languages like Fortran, Basic, Snobol, Assembler, and others. Computer parts are "hardware;" they are terminal interfaces and extension mounting boxes and processors. Programs are "software." Behind computers are programmers, instructors, engineers, mathematicians, scientists, to name a few. But remember, "People have to tell a computer what to do. What they tell it to do is called a program." The computer will talk to you by name, "Congratulations, Noah, you have guessed the word was guitar." And the computer remembers *Inchworm's* every move, no matter how long his trip. Child and parent, we found out more than we knew before we came.

From children I learned about computers and from computers I learned about children. What draws the kids to the computer? For one thing, a kind of push-button hypnotism. Kids are drawn to the instant result, the push-button magic. The dance of letters and numbers and light is another magnet. McLuhan was right. The TV screen is part of the lure—create your own show. It works on the spectator in subtle ways to attract his participation.

We live in an age of remedies. When life goes wrong, and even when it proceeds at its natural slow pace, we feel we must "do" something. The computer "does" it—so we have ready solutions.

The computer intrigues us by its movement; it is not static. It acts, and it works on people by its energetic performance. The child or adult is engaged in activity—press, type, compete, play, experiment, program—instead of being an immobile viewer on the other side of the fence. Part of living in this age is interacting with electronic power. How do we compare with the machine? The Great Machine in the Sky. And then kids have the age-old urge—from flint to cudgel to waterwheel to airplane—to "make it work."

Let's listen to the children, small wizards of electronics, engineering, philosophy, games:

"Does that one work?"

"How's it work?"

"Mine works—watch this."

"Ah, neat-o."

"Look at this thing—it's gone crazy."

"It's a piano. A typewriter!"

"Hey, I beat the computer!"

"I hate these things. They make me dizzy."

"This is getting me crazy."

"What is this? Who pushed the buttons?"

"Dumb computer."

"You guys don't know how to do it. You just gotta press."

"I don't want information. I just wanna turn this thing off."

"Computers can't get mad or cry or laugh or love."

By contrast, what the computer itself says is very low key. "This computer is not a very good tic-tac-toe player. You should be able to beat it sometimes." Take turns with the computer, try to outwit it. Play Simple Simon or bridge. Computers are life forces, to be beaten or not.

The little man was inside the machine. I knew it. It called me by name.

At the Computer Warehouse Store in Boston microcomputers proliferate and look as if they are about to take over the world. The salesman shows me a tiny box, about five by eight inches, in which the switches and parts look as if you need a magnifying glass to see them. Something about angels on the head of a pin dances through my head. They remind me of those ancient carvings inside ivory balls where everything is so minute that you wonder if it had been carved by a race of Lilliputians. Meanwhile the salesman is telling me how ten years ago this particular technology would have filled up the whole store. Now the parts are miniaturized to fit into this box. Under a hundred dollars. With this you can do anything. You can work out a solar heating system for your house, you can help your kid do algebra. You can simulate a moon landing and crash. "A new crater is formed on the moon. Condolences will be sent to your family," says the computer. There is even a Home Computer Chess Meet. Computers play chess with each other, but the owners write the programs in order to compete.

The salesman says these computers are used for college courses to cut out drudgery and promote learning. He

Keyboard Mathematics... try it yourself!
Keyboard Mathematics... try it yourself!

says the software business is mushrooming, and that soon the programs will fill the store like records in stereo shops, and the computers themselves will take their rightful place at the back of the room. Now mostly those who can program have these computers. Soon they will sell their programs in multiples, make a million, and we will all be able to use the computers. Buy a program for feeding the cat, or having a masquerade ball, or driving cross country, or remembering when each kid has to go to the doctor, the dentist, the shoe store. Still, you could learn BASIC programming in one week. A week well spent? Get away from it all for a week?

The store is full of books and magazines about computers. There is a *Computer Gourmet Guide and Cookbook*—I open it. Where are the roast quails on canapes? This looks like gobbledygook to me.

The salesman points to a chip as the smallest, the cheapest. Of course the terminal costs more. But everything is there. Even with bigger support systems, the chip is the same.

This Christmas will see these gadgets run riot. Each Saturday already he is mobbed. Once a week a group comes in to see what's new. The computer store is a new sort of museum. Sales volume has doubled in eight months. The price of components has dropped with miniaturization. Look around you, they are everywhere, inside cars, ovens, TV, fire alarms. They're marching to Pretoria. I wonder if I should buy one....

Computers, I remind myself, are for accuracy. None of your daydreaming inefficiency. Remove drudgery. Oh good, so it's all right to have feelings and fantasies which do not compute. I forgot the question I came with: What can this machine do for *me?*

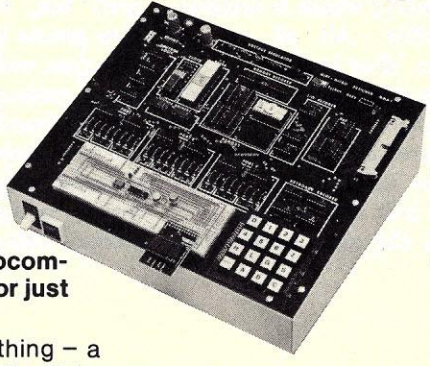
I got so involved with it, I wanted it to respond. Without thinking, I took the Homunculus View; the little man was inside the machine. I knew it. It called me by name. I thought it might talk back. I thought it might help me iron out a few wrinkles in my life. I hate shopping lists, and I heard it could make them. I thought it might help with my Oedipal problems.

I think I got carried away. Too much computer. Leave it in its place, like the food processor. Put in the flour, get out the pie crust, and don't have unrealistic expectations. In an age of increasing reliance on machines, God forbid we should do it by head or hand. Unfortunately, I suddenly remembered that the person runs the machine. Still, there is an interplay between the man and the machine. Each needs the other, like spouses.

Computers improve on our memory and serve us with reams of information. "Information retrieval button," it says. We push. I have a vision of paper spewing forth uncontrollably from the computer, information spilling unceasingly into an empty room or street. Or another of a depressed person "reaching out," trying to make contact, while the computer tapes and programs say, "We have not yet stored loneliness. Computer has no program for your troubles. Sorry. Have a nice afternoon, Captain Cook."

Nice computer, do what we say. But don't bite. ▼

Mini-Micro. Maxi Savings.



The Mini-Micro Designer – a complete microcomputer system for just $830.50!*

Here's the real thing – a microprocessor that takes you right to "real world" situations for about half the price of other systems.

With our hardware, you'll receive the most complete software package in the business. 700 pages of clear instruction, written by Rony, Larsen, Titus – famous for their BUGBOOKS®. Designed to show you how to get your MMD-1 up and working even if you have no prior knowledge of digital electronics.

With our MMD-1 and M/I board combination you'll get all of the interfacing hardware you need, without costly extras.

*Suggested resale price (U.S.A.).

Here's what we pack in for $830.50:
2.5K RAM ... 1.5K PROM (special D-Bug, Monitor and Keyboard Interpreter) ... Audio Cassette Interface ... TTY Interface ... Built-in Keyboard for Control and Data Entry ... Direct Access to latched ports ... Built-in Breadboarding Capability ... Single Step Option ... Monitors for Address and Data Busses. And more.

Best of all, it's on the shelf at your computer store now. Write us for an info-packed brochure and the name of the dealer nearest you.

Dealer inquiries invited.

E&L®

E&L INSTRUMENTS, INC.
61 First Street, Derby, Conn. 06418
(203) 735-8774 Telex No. 96 3536

In our collective mythology we have accumulated several stories about some magical powers in the form of a kind of genie, able to do our bidding (at possibly some cost). The story of Aladdin's lamp is representative of these: the hero finds that if he rubs this old and dirty lamp, out comes the genie to whom Aladdin's every wish is his command. Another story has it that some unfortunate person pulled the stopper off a bottle, only to discover that he had released a very powerful but somewhat malevolent genie. The remainder of the story is concerned with trying to get this genie back into the bottle.

Computers have something in common with these genies: First, they are very powerful and willing to do whatever we ask. Second, they are becoming a problem to society which is unfortunately irreversible. All of us are responsible. Yes, that includes you—you are reading this magazine, aren't you? Unfortunately, unlike the story, we have not figured out how to put the genie back into the bottle. On the other hand, he isn't all that bad.

Let me push the analogy a little: suppose it were you who found the lamp and started to rub it; out comes this awesome genie in a great cloud of smoke, and he says, "Ma-she' elathek wetinathan lak" or some such. Now Aladdin (probably) would have understood him to be asking for a wish to grant, but you do not understand ancient Middle Eastern languages, so you have no idea what he is saying, or how to give him any commands.

One of the major problems in dealing with computers in our time is that of language. We don't speak theirs, and they don't speak ours. More about that later. The other major problem is perhaps better illustrated in the story of the Monkey's Paw. It is not exactly a genie story and I won't go into the gory details (alas, gory they are!) but, in general, the monkey's paw grants its possessor three wishes. The first wish turns out to have disastrous consequences and the second wish is hastily expended in repealing the first, only to find conditions worse than ever; the third wish, with a little forethought, may be used to repeal the second. The point is this: the monkey's paw (or as in our case, the computer) will do exactly what you tell it to do, even if you did not want to tell it to do that.

A computer is a logic machine. This is to be distinguished from a calculator, which does the numerical calculations but all the decisions are made by the operator. It is also to be distinguished from something like the TV ping-pong games which respond in predictable fashions every time you twist the knob or push the button. No, I am talking about the kind of logic that was the preoccupation of the ancient and not-so-ancient philosophers: "All men are mortal; Socrates is a man; therefore Socrates is mortal." This kind of logic is taught in the philosophy departments of the universities, and there is a well-defined set of rules for it. You can build a set of inferences and assertions and the philosophy people call them syllogisms. The computer people call them programs.

The computer is a machine which can make decisions according to the rules of logic. Actually it is not all that

SOFTWARE

The Genie in the Bottle

by Tom Pittman

complicated. These decisions are on the order of, "If this condition is true, then do this; otherwise do that." Besides conditional operations the computer is sometimes also able to do some rudimentary arithmetic (remember, it is not a calculator), and move information around.

What do we mean by moving information around? First, what is "information?" I will not get into information theory at this time—that is a subject for another article. Information is just what you would think: something you know about something or somebody. You might know the color of your eyes, or your cousin's age. You know how much money is in your wallet, and what make of automobile you drive. This is information. To you this information is expressed as "blue," "twenty-four years next September," "thirteen dollars," and "Ford." For a computer we have to make some allowances, because all information stored in the computer is in the form of "yes/no" answers. For example, if there are four possible eye colors (brown, blue, hazel, and green), you can ask two questions:

1. Is the color brown or blue?
2. Is the color brown or hazel?

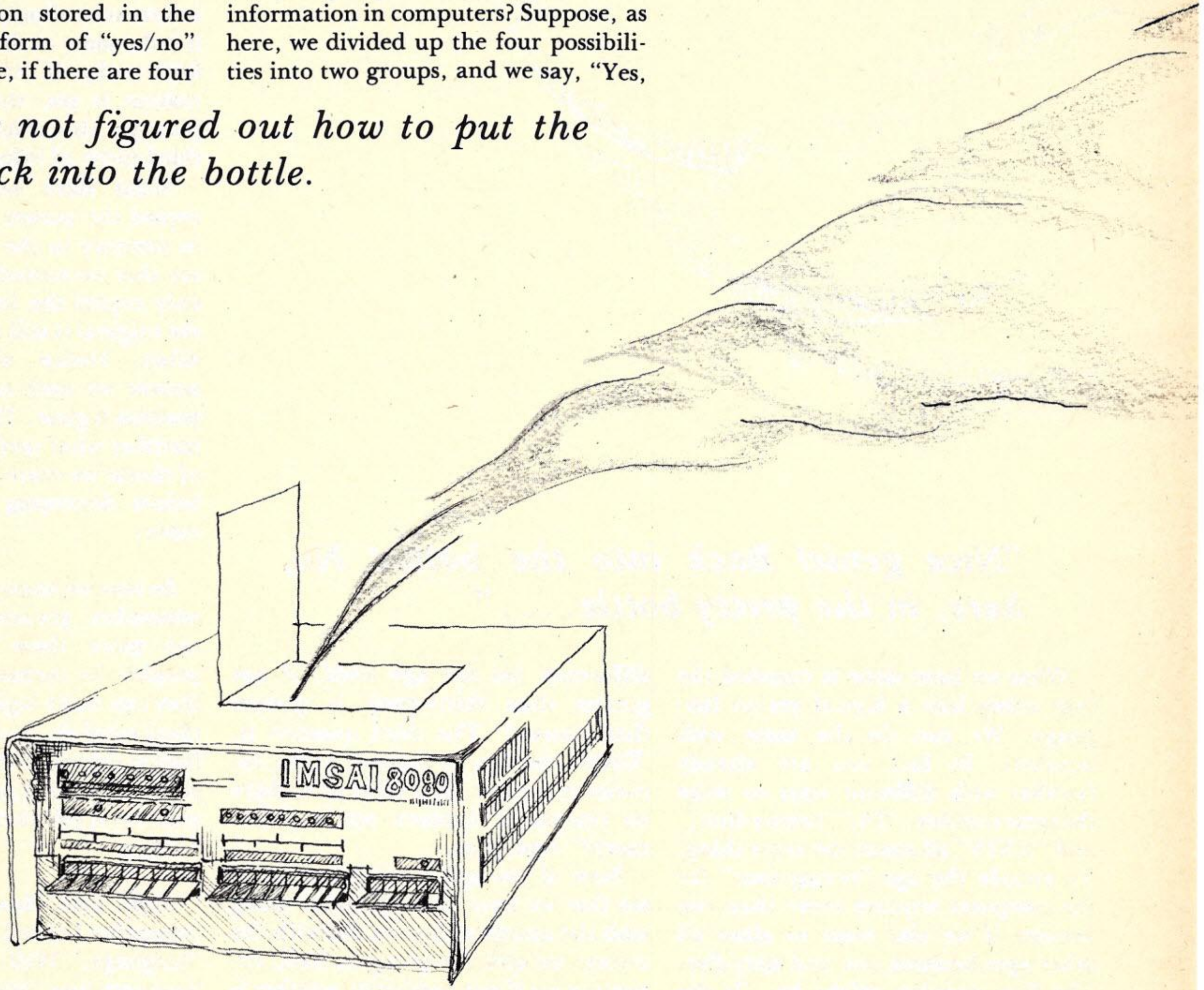
and the answers will tell you exactly what color the eyes really are. In one case, the answers are "yes, no" (blue). For brown the answers would be "yes, yes." You should convince yourself that the two answers are really enough.

One of the major problems with computers is language. We don't speak theirs, and they don't speak ours.

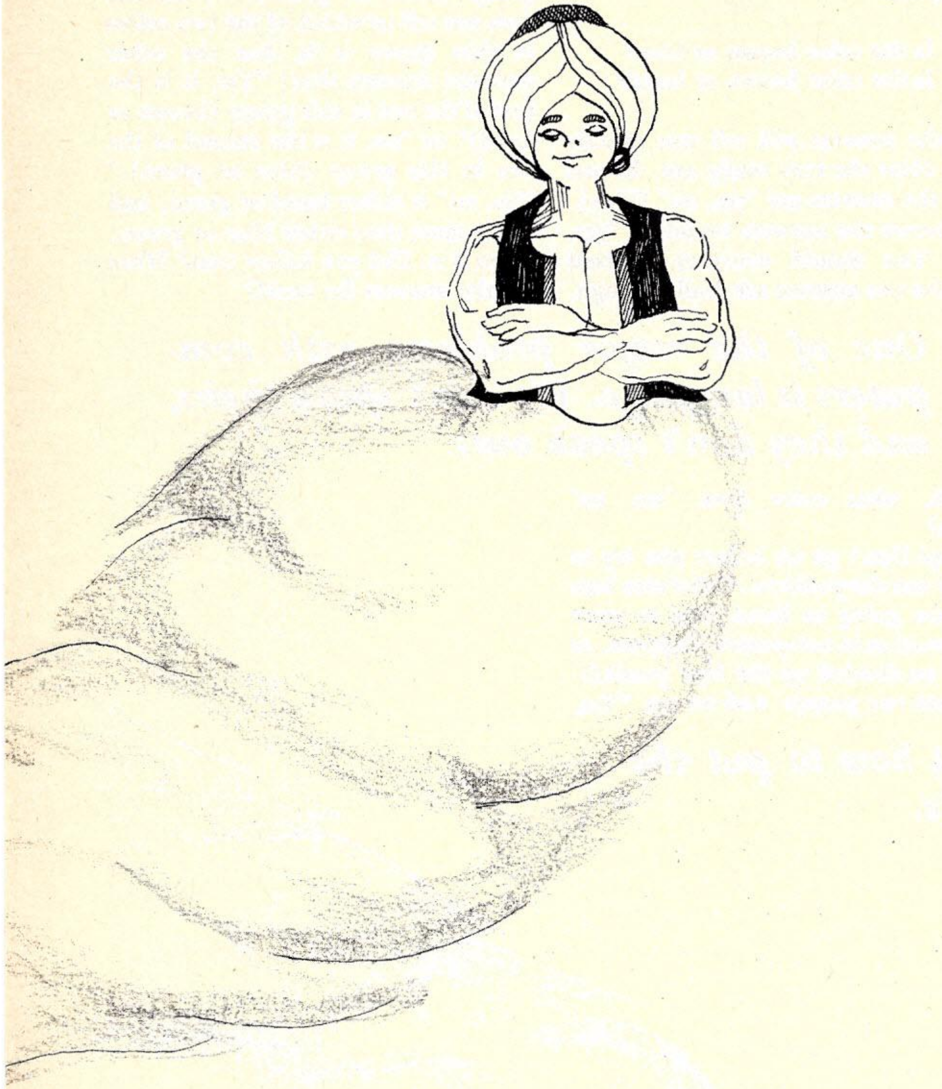
Quick, what color does "no, no" mean?

Stop! Don't go on before you try to figure out the problem. Otherwise how are you going to learn how to store information in computers? Suppose, as here, we divided up the four possibilities into two groups, and we say, "Yes, it is in the left-hand group (brown or blue)" or "No, it is in the right-hand group (hazel or green)." That still does not tell us which of the two colors in that group it is, but the other question answers that: "Yes, it is the first of the two in this group (brown or hazel)" or "no, it is the second of the two in this group (blue or green)." "No, no" is either hazel or green, and at the same time either blue or green; green it is. Did you follow that? What are the answers for hazel?

We have not figured out how to put the genie back into the bottle.



Illustrated by Linda Smythe



"Nice genie! Back into the bottle! No, here, in the pretty bottle. . . ."

What we have done is encoded the four colors into a logical yes-no language. We can do the same with numbers. In fact you are already familiar with different ways to write the same number: "24," "twenty-four," and "XXIV" all mean the same thing. To encode the age "twenty-four" for the computer requires more than two answers if we also want to allow all other ages between one and sixty-five. The first question might be, "Is the age greater than thirty-two?" The second question depends on the first: "If the age is greater than thirty-two, subtract thirty-two, then tell me if the difference (or the age itself, if not greater than thirty-two) is greater than sixteen?" The third question is, "Divide the age by sixteen; is the remainder greater than eight or is there no remainder (answer yes in either case)?" And so on.

Now, if you look carefully, you will see that we have done the same thing with the numbers that we did with the colors: we split the group in half, answering yes if our particular number is on one side, and no if it is on the other. Then we split the half group it occurred in into half again, repeating this until the answer yes or no told us which of two possibilities to choose from. If you write down on a piece of paper all the numbers from one to sixty-five, then keep dividing the list in half, answering yes if the number twenty-four is in the upper half, no if in the lower half, you will get six answers: no, yes, no, yes, yes, yes. And thus you have encoded the number twenty-four.

Why did I go through all this? I was discussing one function of the computer, which is to move information around. The computer is able to remember yeses and noes very easily. These are stored in the computer's memory in some order, so it is proper to say "the first yes-or-no" or "the third yes-or-no" or "the 534th. . . ." And we can meaningfully talk about the 534th piece of information. That being the case, we can also say that the 534th piece of information, ignoring whatever it used to be, should be changed to the answer yes. Or the third piece of information should be changed so that it is the same as the first. That one is a little tricky. If the first piece of information is yes, then make the third piece yes also; if it is no, then make the third piece of information no also.

What have we done? We have copied the answer in the first position in memory to the third position. We say that we moved it, but actually we only copied the information, because the original is still there in the first position. Notice also that whatever answer we used to have in the third position is gone. There is no way to remember what used to be there, unless of course we move it to somewhere else before destroying it with the other move.

So now we know that computers can remember yes-and-no answers, they can move these answers from one position in memory to another, and they can make logical decisions. If you think carefully, you can see what those logical decisions will be based on: the yeses and noes in memory. If the 534th position is yes, then do this; otherwise do that.

Well, now. How does it work? You remember I called the yeses and noes a "language." Well, language it is. We have seen how "blue" can be translated into yes, no. More significantly, we saw how the number twenty-four could be translated into a sequence of yeses and noes. Since the positions in memory are numbered, let us also translate (or encode, if you prefer) each position number into yeses and noes. Following the same steps we did for the number twenty-four, we find that the first position may be encoded as all noes. Position number 534 comes out no, no, yes, no, no, no, no, yes, no, yes, yes, no. Not only can we encode numbers in this strange language of yes-no, we can put the logic that I said the machine was capable of into it.

There is no way to remember what used to be there, unless we move it somewhere else.

When we tell the machine to move one or more pieces of information from one place in the memory, we use a command in some language. Suppose I say, "Move the information from memory position number twenty-seven to memory position number fourteen." That is a command in English. Alas, the genie and the computer do not understand English. So we translate the command into ancient Arabic, Aramaic, or Sumerian, as appropriate for the genie. And for the computer? Binary—the language of yeses and noes. You know how to handle twenty-seven and fourteen; all you need is to translate "Move from ______ to ______." And by golly, there is a special word (or a group of words) in binary which says just that. There are other words in binary which say "If ______ then ______," and so on.

You may notice that I am not telling you what the binary translation is for these more complicated commands. That is because I do not know binary that well myself. But there is a more important reason: every (expletive deleted) computer speaks a different binary language. About the only thing they agree on is the numbers, and they are not all that unanimous on those! "Nice genie! Back into the bottle! No, here, in the pretty bottle. . . ."

Alas, the genie and the computer do not understand English.

Where does that leave us? The genie—er, um, the computer—does not understand English; we (or at least the most sensible of us) do not understand binary. Well, if I had a real genie pop out of a real lamp only to discover that he spoke only Arabic, I might quickly enroll in a night school class in Arabic. But that is dangerous. Maybe there are close distinctions like French "poison" and "poisson," and I might accidentally say "poison" when I mean "fish."

Better I should find someone who will tell me how to say in Arabic, "Speak and understand English!"

Alas, it is not so easy with the computer, but we try anyway. One of the problems is that we cannot tell the computer to do something without telling it explicitly and exactly how to do it. Since we ourselves do not understand exactly how to speak and understand correct English, how can we tell the computer?

This is where software comes in. You know about the hazards of us humans trying to learn binary. This results in what is sometimes called "computer error" but it's really human error. The computer cannot be taught English, so we find some middle ground.

The first cut at a middle ground is what we call "assembly language," which uses the Roman alphabet but is otherwise the same language as the binary the machine speaks. It is about the same as writing Greek or Russian in the Roman alphabet: at least the letters are familiar and we can make out a pronounciation, but we still do not know what it means without learning the language. But it is a step in the right direction.

How does it work? Instead of speaking to the computer in "Binary Absolute" (that is, a long string of yeses and noes), we give it a cryptic string of short words that almost have meaning to real people:

```
LDA X43ZQ
STA GTFUN
CPI BLUE
JZ GOTCHA
```

In this little piece of program, we have

ROMtutorial ROMtutorial

Language (ancient Middle Eastern or otherwise): The means people use to communicate with each other. Language is also a jargon word in computerese which refers to the means by which computers communicate with each other and with humans.

Software: A meaningful term only in distinction to "hardware." In the dry goods (department store) business, hardware deals with plumbing, electrical components, and tools—things that are physically hard, and that might make a dent in the floor if you dropped them. Software, on the other hand, refers to clothing and other goods soft to the touch. In computers the distinction is even more pronounced. Hardware, as before, is the plumbing and electrical parts, the things made out of metal and plastic that if they were big enough would make a dent on the floor when dropped. Software is so soft to the touch that you cannot even feel it. It is the pencil marks on a piece of paper, the holes in a punched card, the magnetization in a tape, the pattern of electrical currents in some transistors. Yet, paradoxically, it is the hardest part of computers for people to understand.

Program: May be thought of in the same sense as when the word is applied to a theatre or a TV program, where it refers to the sequence of events to take place. Programmers are people who make programs. Actually their function is very much like that of their namesakes in the radio and TV business, because they are concerned with making up the program and causing the events to follow it, rather than your concern as the viewer to know what the next event is.

Computer logic: The electronic circuits which perform the individual logical functions. It turns out that the entire computer is made up of these circuits, and computer operations we would not have ordinarily associated with logical decisions are, on a microscopic electrical level, nothing more than that.

Encoded: A fancy word meaning "put into code." The code here is a language of yeses and noes, but it could have been dots and dashes like in Morse code; the idea is the same. We have another word, "decode," which means getting out the meaning (blue or green, or whatever) from the code, whatever the code is.

presumably given some memory location a name, "X43ZQ," a different memory position is named "GTFUN," and the first two lines are commanding the computer to move whatever yeses and noes are in the location that we named X43ZQ to the location that we named GTFUN. If the information we moved is exactly the same as the pattern of yeses and noes that we have named "BLUE" then we want the computer to do the part of the program that we labeled "GOTCHA" and otherwise it is to something else (unspecified). Note that there is a lot of freedom in assigning names and labels to things. We can choose words that are hopefully more meaningful to us than "534" or "27." While it is very easy to write down 24 when I mean 27, it is a little less likely to write "BLUE" when I mean "GREEN" so it is less likely that we will make those human bloopers that the uninformed call computer errors.

> *Push the appropriately labeled key, and out the other side come seven electrical yeses and noes.*

In an assembly language some of the names are already defined; such were "LDA," "STA," "CPI," and "JZ" in the example. These are supposedly easily-remembered names for the verbs in the computer language. Each line is a sentence consisting of a verb and an object. We get to name the nouns; someone else has already named the verbs and the adverbs and the pronouns. Beginning to sound like a real language? Of course we don't call them verbs, nouns, adverbs, etc. in computerese; we call them "Opcodes," "Labels," "Modifiers," etc., but the meaning is the same.

I have not yet explained how the computer understands this new language. Remember, it speaks only binary. What we do is find somebody who speaks Arabic—er, ah, binary—and get him to tell the computer (how) to understand assembly language. And this has been done, so let us metaphorically look over his shoulder to see what he did.

First we have to worry about the alphabet. Just like the Arabic understood by the genie, the language understood by the computer has a different alphabet (yeses and noes). So we set up a correspondence table, and build a piece of hardware to make the appropriate substitutions. The correspondence table is called "ASCII" (pronounced "ass-key"). Each letter, digit, or special character is assigned a unique seven-bit code (seven yeses and noes) in much the same way we assigned codes to the numerical positions of memory: each character is given a numerical value, then the binary equivalent for that number is used. The letter "A," for example, has the numerical equivalent of 65, "B" is 66, and so on. The lower case letters are also encoded: "a" is 97, "b" is 98, etc. The decimal digits are included in the code: "0" is 48, "1" is 49, and so on, to "9," which is 57. A period is 46, a comma, 44. Since we need some way to tell the computer about the spaces between words, we even have a special code for a space: 32. There are also special codes to mark the end of a line (13), to indicate a tab operation (9) or a backspace (8) (just like a typewriter; you will see why), and even to ring a bell (7). And there are two special codes which mean "nothing at all:" 0 and 127. These last two codes are the "all yeses" and "all noes" codes and are often used to take up space in memory without signifying anything. The special piece of hardware which makes this conversion is called an "ASCII Keyboard." You push some appropriately labeled key, and out the other side come seven electrical yeses and noes corresponding to the code for the key you pushed.

Now we know how to represent the assembly language to the computer. It will see some 32s (or rather the yes-and-no combinations which represent 32s); these are the spaces at the beginning of the first line. Then comes 76, 68, 65, another 32, followed by 88, 52, 51, 90, 81, and a 13 to end the first line. Remember each one of these numbers is actually seven yeses and noes, and not a number at all.

That was the easy part. The hard part is telling the computer what to do with all this data. This is done with a program. Since the computer only understands Binary Absolute, the program is given to the computer in Binary Absolute (like maybe by reading

ROMtutorial ROMtutorial

Memory: The ability of the machines to remember things, specifically those ubiquitous yeses and noes. A "4K bit" memory has room in it to remember 4,096 different yeses or noes. The "K" is derived from the Greek word "khilioi" meaning a thousand, which has been adapted in our scientific culture in the prefix, "kilo-" also meaning a thousand. Some pedants insist that K means 1024, and you will sometimes see the term "64K" which is intended to mean "65,536."

Binary language: We could abbreviate our yeses and noes as Y and N, so that 27 comes out NYYNYY and 14 comes out NNYYYN, or as is more common, we write "1" for Yes, and "0" for No, and the words look like 011011 and 001110. Each one or zero (yes or no) is called a "bit." Some people feel this is a contraction of the words, "binary digit," but I think it is descriptive of the fact that each one or zero contains just a little "bit" of information, just the answer to one yes-or-no question. Exactly as English words are a collection of (roman alphabet) letters, so computer words are a collection of bits, of ones and zeroes. The difference is that in a given computer, all the words are the same size.

String: Another borrowed word in computerese. It has the same sense as in the phrase, "string of beads." In other words, we have a bunch of objects of similar size and composition (though not necessarily the same color or shape), which are lined up in single file, and held in place by some thread. For beads the thread is the string itself; in computer strings the thread is conceptual. It may be that the objects are in consecutive memory positions, or that they are presented to the computer in sequential order.

Label: Normally we think of a label as, say, a piece of gummed paper which is stuck to an object, with some words written on the paper to tell us what the object is, like the label on a jar of jam, that has the word "strawberry" on it. Actually we are more concerned about the words than the gummed paper, so a label in a program is a word which is somehow attached to a part of the program. In some sense it tells us (or the computer) something about that part of the program.

the program in off a cassette). The program is called an assembler, and when the computer is running the assembler it understands assembly language.

I say *"It* understands assembly language." At this point it becomes extremely difficult to pin down whether it is the computer or the program which is actually doing the understanding. In point of fact, neither the program nor the computer by itself is capable of understanding anything. It is rather the combined work of the two. Some people (notably Dr. Weizenbaum, whose thinking on the matter you can read elsewhere in this issue) would insist that even computers with programs do not "understand" anything, and to some extent I suppose he is right. By now, however, I hope you have noticed that we computerists go in heavily for metaphors.

When I say that a computer or program "understands" some piece of information, I mean that it is able to take some meaningful action on the basis of the value or significance of that information, and that if I had given the same information to a human being, I would say that person "understands" it if the same action is taken. The difference is that while a person may perceive and grasp the concepts relating to the information, the computer is only doing what the program tells it to do. If we had given the human the same instructions, the same action could occur without any understanding (in the human sense).

It becomes extremely difficult to pin down whether it is the computer or the program which is actually doing the understanding.

The function of the assembler, given that it (in some sense) understands the assembly language program, is to translate that program into machine language (that is, Binary Absolute). When it sees "LDA" it will convert that into a binary word which represents that particular operation in that computer. "X43ZQ" it will recognize as one of the programmer-defined labels, and substitute for it the numerical memory position (or rather the binary encoding of the numerical position) which that label represents. The assembler may store these coded commands directly in memory, or it may send them out to a cassette or some other device, so that when you are ready to use the program you can load it into memory and run it.

I hope I do not need to tell you that this is not the ultimate in man-machine communications. Assembly language is hard to learn, easy to make mistakes in, and very tedious to write. We continue to hope for something better. Enter BASIC.

BASIC was developed at Dartmouth College by Messrs. Kemeny and Kurtz as a language to teach programming. I suspect they thought of it as a "simple-minded FORTRAN." Anyway, the language is fairly easy to learn (compared to assembly language), and it is not as tedious to write programs in it. As a computer language it has achieved great popularity because it is taught in the schools. While some people claim that BASIC is "like English," I can assure you that is false. It is an algebraic language, which is mathematical in nature. In BASIC it is easy to talk about your cousin's age or the amount of money in your wallet. But if you are concerned about the color of your eyes or the kind of car you drive, you are almost forced to convert these into numbers. So to some extent you have the same problem that we saw in machine language—we have to translate our commands before the computer understands them—although this is much less of a problem in BASIC than in machine language or assembly language.

If BASIC is so much easier for us, does that mean it is harder for the computer? You betcha. In general it takes about twice as many instructions to tell the computer how to understand BASIC as to tell it about assembly language.

There are other problems, too. When you write in assembly or machine language, every command you write is exactly one command for the machine. And it will execute these commands at the rate of about 200,000 each second. In BASIC however, each command or line of the program may result in hundreds or thousands of machine instructions. To do some task in BASIC may take considerably longer than the same task in assembly. And there are

ROMtutorial ROMtutorial

Assemblers (various): Usually an assembler accepts programs for the computer the assembler is running on; this is called a "resident-" or "self-assembler." However when a new computer comes out there does not exist an assembler that runs resident. Rather than writing that first assembler in Binary Absolute (Yecch!), we usually write the first assembler to run on a different computer, one which already has a language processor like an assembler or a BASIC interpreter. This is called a "cross-assembler."

BASIC: Those of you with acronymania may appreciate that BASIC is supposed to stand for "Beginner's All-purpose Symbolic Instruction Code" but I think the acronym came later.

FORTRAN: An early programming language which has achieved widespread use in larger computers. Its name is a contraction of "Formula Translator." It is in some ways a better language than BASIC, but it is also more complicated and harder to learn. It is also harder to squeeze into a small computer, but that is a different story.

ROMtutorial ROMtutorial

Floating-point numbers: Numbers stored in the computer in scientific notation, that is, with the decimal (or binary) point "floating" at the front of the number. To make sure we do not forget where the binary point really belongs, we keep a second number which indicates how many places over it belongs (in one direction or the other). To see how this works, go borrow somebody's fancy calculator and multiply 100,000,000x100,000,000. The answer should be 10,000,000,000, 000,000, but they do not have enough digits to show all that, so instead you will see "1.000000 + 16" which means that the decimal point needs to be moved to the right sixteen places before the answer is correct.

Lunar Lander: A game in which you pretend to pilot a spaceship down to the surface of the moon. You specify how much fuel to burn and for how long, and the computer calculates your speed and distance from the moon. It involves several complex mathematical equations.

some things you simply cannot do as easily in BASIC as in machine language.

What it boils down to is this: BASIC is a much better language to write programs in if what you want to do is easily done in BASIC. BASIC is probably still a better language to write in if what you want done can be done at all in BASIC. Otherwise, well, back to the old rock pile (assembly language).

But we have a dilemma: I said that a computer is not an arithmetic machine (a calculator), but a logic machine. Now I say that it can understand and perform BASIC language programs, and that BASIC is an algebraic language. What do I mean? It is true that you can say in BASIC

```
LET A3 = X + D1 (X-4.3)
```

and that it will subtract 4.3 from the number named "X," multiply the difference times the number named "D1," add the product and the number "X," and then label the sum "A3." It is also true that most of the computers that we as amateurs can afford not only do not know how to multiply, they cannot even add or subtract fractions. The catch is the software. Somewhere along the line the computer has been told how to convert the character string "4.3" into something resembling a binary number. Usually it uses what we call "scientific notation," where the number is represented inside the computer (i.e. stored in the memory) as some binary pattern corresponding to ".43x10," or, more likely, ".1000101010101010, 0011." The number labeled "X" has a similar representation. The computer knows how to subtract one small number from another small number, but these are not small enough. What do we do? We activate a special program which tells the computer how to line up the binary points in large numbers and subtract with borrowing. The carries and borrows are the same thing they used to teach us in grammar school.

> *Assembly language is hard to learn, easy to make mistakes in, and very tedious to write.*

When the computer needs to multiply, another part of the program is activated, and its sole function is to tell the computer how to do long multiplication. There is another part of the program that tells how to do long division. Only the program is incredibly simplified. If you want to multiply 4x3, you don't look it up in the times table that we all memorized as children. I suppose the computer could "memorize" a times table, and I once wrote a multiply program that did it that way, but usually we just tell the computer to add three, four times:

$$4x3 = 12 ; 3 + 3 + 3 + 3 = 12$$

All of this is done in software. Sure, the big computers have hardware to do floating-point arithmetic, multiplication and division, and all the other number-crunching operations we have come to know and love. If we want to do it at all, we do it with software.

But let me tell you, most of the things we can use the computer for do not need all this number crunching. The last time I did something in floating-point arithmetic on my computer was some two years ago, to enter a programming contest (I won). Look inside a program sometime (not BASIC). Count how many arithmetic instructions you see, compared to the total number of instructions. Even programs with a heavy computational flavor like *Lunar Lander* spend only a small percentage of the total code in arithmetic. Why? Because the computer is a logic machine, not a calculator. Now, I excepted BASIC above, because BASIC is a numerical language: you find yourself doing things with arithmetic instead of logic. I think that is too bad, but BASIC is available, and better languages are like the weather—lots of talk, but nothing is being done.

I started this article comparing computers—or rather the software that makes computers something more than a pile of lights and switches—to the mythical genie. I talked about some of the problems in having your own private genie, but I have not said much about the wonders he can perform for *you.* There is a lot of talk about what small computers can do; so I will restrict myself to what is actually being done.

About two years ago I was designing an improved TVT for my system, and I wanted to use a new fifty-dollar part. Rather than spend the money and find out I had made some disastrous mistake, I wrote a computer program to simulate the action of the circuit. That is, each component of the circuit was translated into yeses and noes in my computer's memory, and I gave it a program which told the computer how to pretend that it was the new circuit. One full interlaced scan frame on a TV set is 1/30th of a second, but it took my program something over eight hours to simulate it. What with making corrections to the program (nobody writes a program that runs perfectly the first time!) and modifying circuit parameters, I had the computer running non-stop for eight days. But when I wired up the circuit it worked almost perfectly right off (except for two small glitches I had not noticed in the simulations, but which did not affect the performance).

I am running a small business, selling "Tiny BASIC" interpreter programs to owners of 6800s, 6502s, and 1802s. For each order that comes in, we type in the name and address to the computer. Every time we have a new product release, the names are sorted by zip code and the computer prints out mailing labels. Otherwise we keep the list sorted alphabetically to make it easy to find a particular customer (who may be waiting on the phone). We use the same programs to maintain the local computer club mailing list.

When a customer buys a program, we send out either a paper tape or a hexadecimal listing (human readable form for people who do not have hardware to accept paper tape) of the program machine language. Every tape (or listing) has a unique serial number attached to it by the program that punches (or prints) it. The tape is punched, then read back into the computer to compare it against the original. If they do not match, the computer defaces the tape and punches another, so that the customer is assured of a perfect tape. "It's all done with mirrors, uh, er, I mean software."

I am not much for playing games on my computer, but I wrote one which played a mean game of Tic-Tac-Toe (you against the computer). I thought the program was unbeatable, until my wife played it and won. Score one for the genie.

More recently, I wanted to write a program in a new computer language (oh, how I hate to learn new languages!), and I needed an assembler. Well, actually I could have written the program in Binary Absolute, but I estimated that would take two or three weeks to get it going, what with correcting all the mistakes I would have made. So I knew I needed an assembler. I had three choices:

1. I could modify an existing assembler (that I had not looked at in years) to work for the new machine. Estimate two weeks of fiddling around.
2. I could write a new assembler in assembly language (different machine). Estimate three or more weeks to get it working right.
3. I could write the new assembler in Tiny BASIC. Now I should tell you that BASIC, and especially Tiny BASIC, is not a suitable language for writing assemblers. But I chose this route, and had the assembler running in two days. It was incredibly slow, but I put it on in the evening and came back for the output the next day. Total elapsed time: one and one-half weeks.

Moral of the story (all these genie stories have morals): Use the best language you can get; anything is better than assembly language (except possibly machine language, which is the worst of all). The more of the job you can let the software do the better off you are. ▼

Most of the things we can use the computer for do not need all this number crunching.

ROMtutorial ROMtutorial

TVT: Short for "TV Typewriter," a term given by Don Lancaster to an inexpensive circuit to put computer text on the screen of a TV set. This is the reverse function from the ASCII keyboard.

Interpreter (for BASIC or any other computer language): The program that tells the computer how to understand the BASIC language, just as an interpreter follows the President around in foreign countries, telling him what the people he is visiting are saying.

Computer games: Programs which take input from the keyboard (understanding it as moves made by the human player) and make various logical decisions (*Star Trek* is a popular theme for computer games, partly I suppose because the decisions are, in the word of Spock, "logical"), then display the computer's move and/or the current state of the game on a TV set or printer.

POSTSCRIPT

TINY BASIC is not as good as any regular BASIC. If you can get a regular BASIC for your computer, by all means, use it. This is true now for the 8080 and the 6800. Some 6502s now come with a BASIC option. Buy it, it's well worth it. Otherwise send a stamped self-addressed envelope to

ITTY BITTY COMPUTERS
P.O. Box 23189
San Jose, CA 95153

and we will tell you all about our TINY BASIC.

Your Computer or Your Wife?

by Susan Gilpatrick

"If you can just convince my wife. . . ." How many times have I heard that? I'm not certain, but it has been often enough to make me stop and think about it. Why are so many wives so dead set against a personal computer? (To clarify one point—I am not being sexist; to date in our computer store we have not had one woman say that while *she* would like a computer, her husband couldn't be persuaded to let her buy one.)

Why are women so dead set against having a computer system at home?

Justification. The personal computer must be justified. It is as if the computer were a useless toy. TV sets and stereo systems are purchased without any discussion of what they will be used for. Obviously, they are for entertainment and possibly some education. Yet, when a personal computer system is discussed, it is paramount that it do something "useful."

Actually, when you buy a personal computer, you are making an investment in a product which can give an almost unlimited return. Its uses in the home or business are limited only by the user's imagination, initiative, and expertise. It is an excellent source of entertainment, but it also does the jobs that a layman would expect a computer to do and is, in addition, a superb educational device.

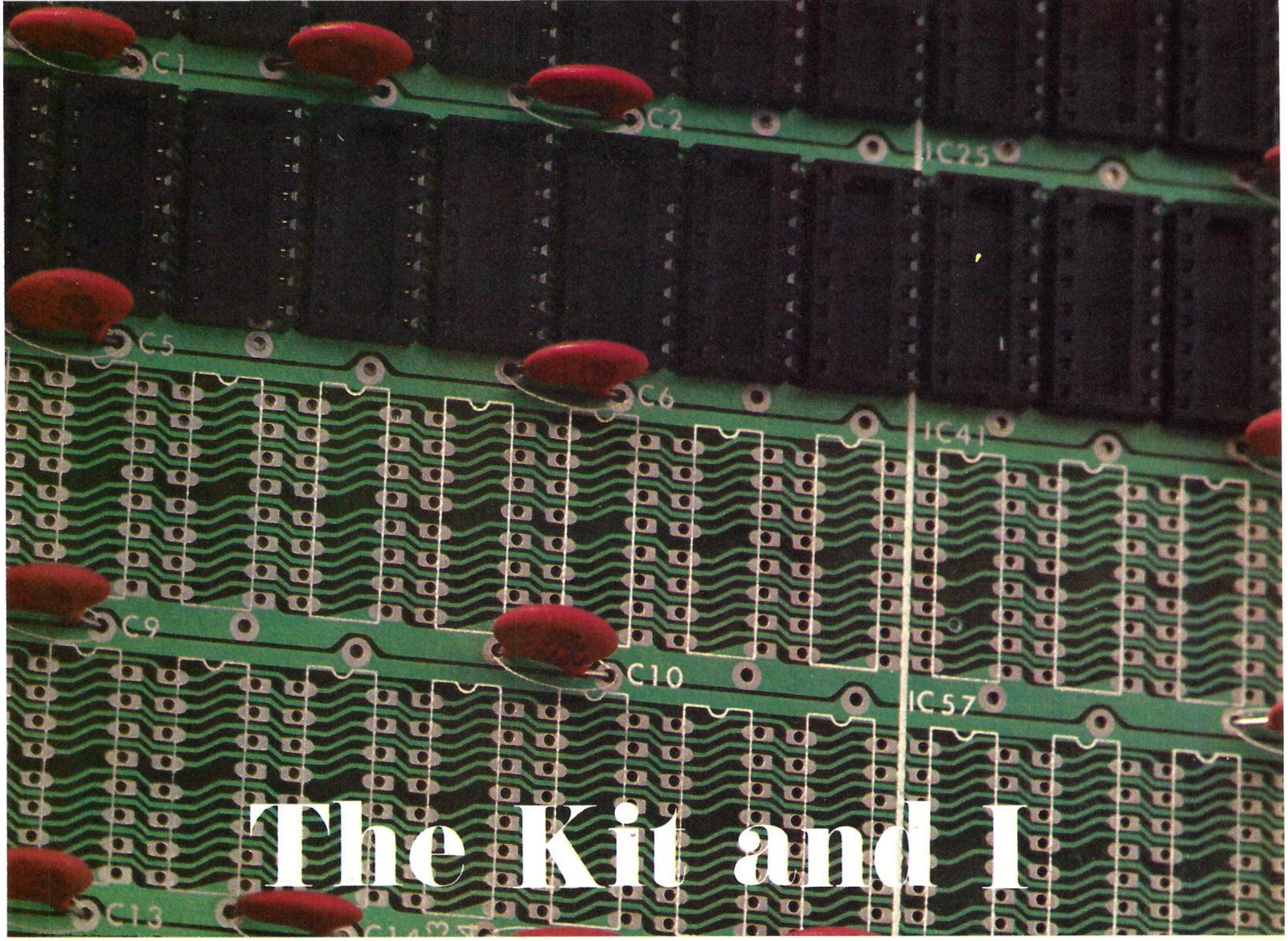
As an entertainment center, the computer can provide the user with games to play—games for one player, games for two or more players, even games against the computer. These games may be from published program listings or they may be developed by the user. In which case developing them is an education in and of itself. They can be adaptations of games that are already marketed or they may be designed specifically for computers. A computer can also be used in conjunction with other leisure activities in such tasks as controlling model railroads or road-racing sets.

Perhaps one of the most exciting uses of the home computer is for education. Math problems are suddenly full of suspense for a child when they flash up on a CRT or are typed out magically on a printer. Multiple-choice or fill-in questions are fun when the computer asks them. Parents may quickly find that, with the right program, it is sometimes difficult to get the child to stop studying. Aside from helping a child with his school studies, the computer opens up a whole new world for the student. Programming is an excellent method for learning the principles of logic. Logical thought, coupled with the precision and orderliness that programming demands, is a way of thinking that will be carried over into other facets of life and learning. As more and more of our world becomes involved with computerization, programming techniques and skills will increase in value.

Of course, the personal computer can be used for home or business bookkeeping, tax records, Christmas card lists, even menu planning. With some additional hardware, the computer can act as a security system or monitor the furnace, water pump, and air conditioning. As time goes on, many of our homes and appliances will either use their own microprocessor or be tied in to one.

If all this is true, then why are women so dead set against having a computer system in their homes? Many of those wives have known about computers only through their husbands. And, all too often, this has been anything but pleasant. It must be that people involved in data processing are in a certain class by themselves. They just don't understand why wives are upset about strange hours, an incomprehensible vocabulary, and the strange attachment their husbands have to a machine. Computers have, in the past, sometimes meant husbands who went to work in the middle of the night (to get machine time), or ridiculous conversations "in computerese" on the rare occasions when those husbands were around. But the most difficult thing to face was that a *machine* was making these demands. It was not even a fanatical boss who was issuing the orders. Only because of the strange relationship of man and computer, do these problems arise.

People are going to have to realize that a computer in the home does not have to cause aggravation and more problems. In fact, nowadays it can be used as an opportunity to close the gap. Some of the jargon will, of course, creep into the family's vocabulary soon, so the computer will not be a remote and foreign entity. It will begin to take its place in the home alongside our TVs and stereos, our blenders and food processors. If husbands can learn to show some understanding and can get computers to take over some of their wives' dreaded chores, then the personal computer will be a welcome addition—rather than the enemy. ▼



The Kit and I

by Richard W. Langer

It was an unusually cold stormy May day when the UPS man pulled up the country lane leading to our farm. Unceremoniously he handed me a package the size of two shoe boxes. It felt awfully light for a computer, I thought, scribbling the old Hancock on the receipt form.

I took the package inside, hardly able to wait for a knife with which to cut the strapping tape. Three quick slashes and the box opened, exposing a bag full of electronics parts and a green instruction manual. Seems the 8K memory module had arrived. The computer itself was still somewhere in transit.

Stirring up a few charred logs in the fireplace to keep the chill away, I curled up in a chair and turned to page one of the instruction manual, entitled *8KRA Static Read/Write Memory Module Assembly and Test Instructions.* Now, for someone whose only previous contact with electricity came from rewiring an old electric stove in such a fashion that it blew either fuses or the frypan right off the top burner, the days to follow were to prove quite an experience.

That first evening I contented myself with reading the manual—several times. I supposed it might make sense to wait for the computer itself to arrive before actually starting the assembly project. But in the end I couldn't restrain myself. Any way one looked at it, I told myself, the computer was going to need memory.

With all the manual's warnings about static electricity and synthetic-fiber clothes, I decided to run right out first thing the following morning and buy some cotton shirts and underwear. An over-reaction, no doubt, but having gotten into the drip-dry habit, this seemed as good a time as any to get out of it. Susan, who really doesn't have time for all that ironing, suggested I assemble the kit in my pyjamas. The only time I had to work on the project was late at night—and the sleepwear was one hundred percent cotton, she assured me. I

Photographs by Thomas Hall

The sorted parts hang labeled and taped to the wall waiting for Langer to complete testing of the board.

settled for forty percent cotton and kept my shirt on.

Fashions aside, definitely I would need an ohm meter. Our electronics tool chest was limited so far to some twenty-amp fuses, some thirty-amp fuses, and some wire cutters I had used for taking down an electric fence on the west forty. Popping over to the local Radio Shack, I picked up their super-duper-deluxe-model volt-ohm meter. It had more knobs on it than my stereo. No doubt they'd all do something useful sooner or later on the job.

While I was there, I also procured more diminutive wire cutters, needle-nosed pliers, and a twenty-five-watt soldering iron. The manual had suggested a "controlled heat" soldering iron in one spot and a plain twenty-five-watt one in another. The salesman assured me a regular one would do, I suspect because they were out of controlled heat models. I settled instead of selected.

Armed to the teeth for tackling my project, I returned home to answer that age-old question (about two years age old, actually), "can the average Richard, who's done little more in the world of electronics than change a flashlight bulb, assemble a computer kit?" For starters, the answer looked like a good solid "well, probably."

First there was the little matter of checking that all the parts were indeed there. Not such a simple task if one doesn't even know what a diode looks like. Still, there were pictures, which, crosschecked against parts numbers, began to make sense.

At first I noticed a lot of missing parts. By the third time I'd gone over the checklist I'd found everything. Well, almost everything. I still had only ten small resistors instead of the eleven specified. I also had some extra goodies. As it turned out, these ended up on the board anyhow (see the later pages of the instruction book for how that happens).

The big guesses were the right angle molex connector and the augat pins. I'd never met such animals before. But by the process of elimination—and a little telltale *MX* on the side of a brown doojiggy with right angle pieces and a matching thing to plug it into—I assumed I'd found the molex connector. That left only one part, a little row of pins. Augat pins, my dear Watson. Elapsed time about forty minutes. Now, on to the assembling.

Or so I thought. As it turned out, I had to check the board for shorts first. That meant unpacking my trusty volt-ohm meter. The instructions before use read, "Remove the case back and install the batteries." Of course they didn't tell you how to remove the back, and there was nothing on the meter to indicate a method either. There were three deeply recessed screws, however. I went for a Phillips-head screwdriver.

The front came off instead of the back. Nevertheless the batteries fitted into place without any trouble. I zipped it all together again.

The needle did not rest at zero when the control switch was at the *off* position. For this situation there were remedial instructions. One was to adjust the needle by means of the plastic screw at the lower center of the meter face. Reasonable enough. But a *plastic* screw? I went for a regular screwdriver, musing meanwhile on the fact that all the Radio Shack equipment seemed to be made in Korea, while the chips used by Processor Technology came from Singapore. Whatever happened to the fabled American electronics industry?

Once I had the needle set so it rested on zero, I began to test the circuit board. To do so, I had to figure out what to set the volt-ohm meter on. Obviously I was testing for some kind of movement of the needle. First I instinctively set the meter on the DC side. Nothing happened. That was as it should be. But how did I know the meter knob was set to the right setting?

Thinking about it for a minute, I

That left only one part, a little row of pins. Augat pins, my dear Watson.

reached the obvious conclusion that I was testing for conductivity. When the probes were touched to a screwdriver, the needle refused to move. Now even I know that a metal screwdriver conducts electricity, so obviously the dial was in the wrong position. Turning it to various other points, I eventually ended up on the ohm side. The needle jumped. Maybe that was what I wanted? Sure enough, upon reading the instructions again, I came across "Using an ohm meter on its lowest scale. . . ." Obviously I'd better read more carefully.

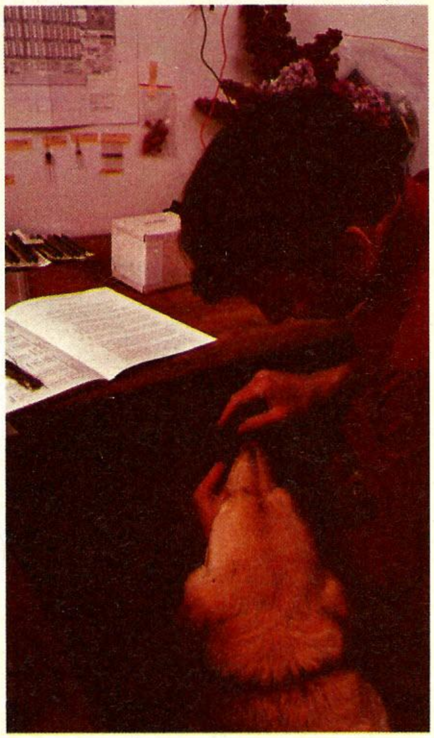
I ran through the eight-volt bus test, the five-volt bus test, and the RAM area test. The instructions I followed to the letter (unfortunately not at first to the plus sign—back one step, forward two). Once I discovered which pinholes to test, everything checked out. Okay. The board passed inspection. The rest was up to me.

The second step was to install the heat sink. Now I admit the kit had looked like it contained everything but—and perhaps including—the kitchen sink. Still, I was surprised to learn that the manufacturer admitted it. Checking my trusty dictionary, I discovered that a heat sink collected heat from hot parts, keeping them from burning out. Clever, these Japanese.

The sink was supposed to sit on the board so that the triangles of the sink matched the three mounting holes on the board. This should have been easy enough to arrange. Unfortunately, there weren't any triangles on the board. With some protracted squinting, however, the sets of holes in the board could be conceived of as triangles. I so conceived of them.

Setting the screws, lock washers, and nuts required no more than three hands, since there was at least 1/32 of an inch between the heat-dissipating wings of the sink and the sides of the nuts.

It was at approximately this stage of the game that I decided I'd better learn soldering. From an electronics junk shop for skilled scavengers on Canal Street in New York I picked up an old NCR board. I was going to practice by removing some of the old parts that I now recognized as ICs and resistors, not to mention those metal can transistors pictured in the assembly manual for Processor Tech's 8KRA memory module. The transistors I couldn't find in my kit because, as a phone call to Processor informed me, they weren't supposed to be found there. Seems the original design called for them, the final kit didn't, and no one had gotten around to removing the pictures. Oh well, as I said, I had purchased the board to practice soldering.



Faithful Christina checks it out. "Hold it. You can't put the sockets on yet. Read the instructions over."

Unfortunately, I couldn't wait to put the real thing together. The NCR board still sits on the shelf.

The needle jumped. Maybe that was what I wanted?

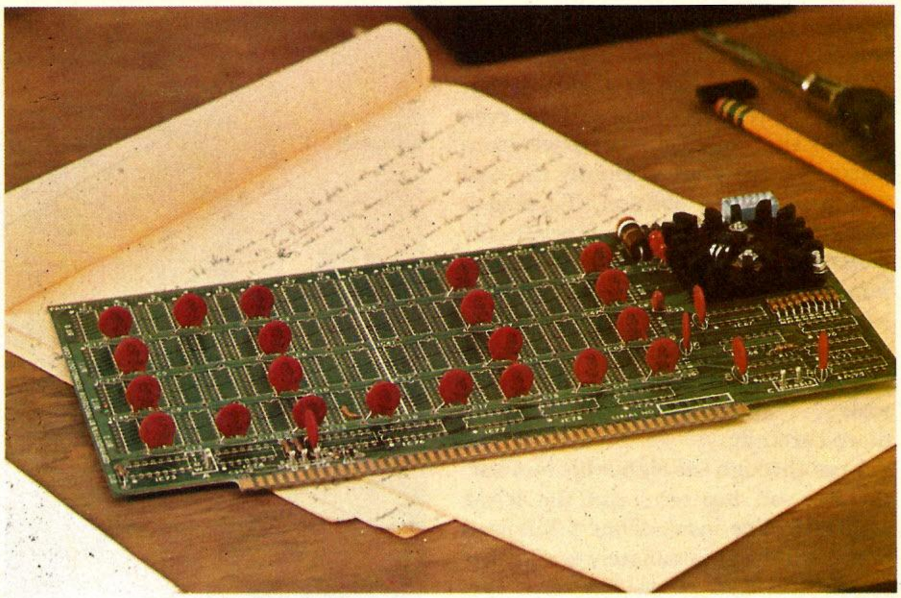
Starting with IC 65 and IC 66, which reposed in the heat sink and had such long leads I didn't worry about toasting them, I moved right along. I was pleasantly surprised to see I could do it. Actually it wasn't difficult at all. Not only did I not lift any of the metal traces the manual warned me about, but the solder flowed into place evenly, looking almost as neat as the professional job on the shelved NCR board.

Clipping off the excess leads was something else again. Metal splinters flew all over the place. Contemplating attaching a magnet to the clippers, I passed up the temptation. I wasn't at all sure the magnet wouldn't affect the ICs. Still, there must be a better way to do it than to suffer the slings and arrows of outrageous leads.

Next came the installation of the male molex right angle connector. Since the heat sink was higher than the connector, one couldn't just put it in on the board and turn the board upside down to do the soldering; the connector would simply fall out. I taped it in place with masking tape. That worked fine. Now I was faced with soldering the five pins in place without forming a soldering bridge between them; they were closely spaced. It was a warm evening this time, but that wasn't the reason I was sweating so much.

Again things went smoothly. So much so, in fact, that I decided not to wipe off the tip of the soldering iron after each joint as the manual instructed me to do. By the fourth pin I'd really fluxed things up. After spending ten minutes cleaning up the excess flux, it was back to following directions for me.

The next step called for installing four diodes at the top of the heat sink.



Notes were kept on what happened when. Always helpful for tracking down mistakes.

Arrowheads on the board indicated precisely which way they were supposed to face. Unfortunately there were no equivalent arrowheads on the diodes themselves. Taking logic into my own hands once more, I decided that the dark band on the diode was somehow indicative of the current flow or whatever, and that it should be the direction in which the arrow pointed.

After I had installed all the diodes, I reread the paragraph about them. Lo and behold, the last sentence was "Position D1, D3, and D4 so that their dark band marks (cathode) are at the bottom, and position D2 so that its dark band is at the top." Now I *know* that sentence was not there when I read the instructions the first time. I *know* it. Really I do. Well maybe....

The agenda at this point was to install the disc capacitors, those orange twin-stick lollipops of which there was a whole minibagful inside the larger bag. They were made of cheap clay or something like it, for as I started to bend the leads, the edges crumbled off a bit. I held my breath, expecting at any time to end up with a handful of dust.

It didn't seem to make any difference which way the lollipops faced, so I decided to line them all up with the imprint facing the bottom of the board. Just for symmetry. Besides, why have half of them in right if I could be all wrong?

By the time I had soldered about a third of the twenty-six capacitors, I was beginning to feel like a soldering pro. My pace was really stepping up. When I finished this group, I held the board up and viewed it from the edge. Actually the soldering looked pretty good. There were a few diminutive foil-wrapper Hershey kisses where overconfidence had stepped in once more.

Now for the resistors. The big one was a thirty-nine-ohm two-watt unit called R1 on the board. The instructions read not to install it unless battery standby power was to be used. Since we live on a farm—I can hear the rooster crowing as I work on the memory board, and let me tell you that's one screwed-up rooster, since as I write this it's almost two in the morning—and since our power company is as screwed up as our rooster, I decided on the battery standby. That left me with only an aesthetic decision. At least I hope that's all it was. The resistor has orange, white, black, and gold bands on it. I reasoned that a resistor resists current whichever way it faces, and I decided to put it with the orange band on top, where it would artistically balance the orange capacitor just below it.

It was when I began installing the smaller resistors, the 1.5K or 2.2K ones, that I realized my worries about being one short were for naught. Although the parts list called for eleven and I had received only ten, the board had room for only ten, so everything presumably was all right. It encouraged me that they couldn't count either. By now, of course, I had realized there's many a slip 'twixt design, manual, and final kit. Not all changes were accounted for along the way, hence the discrepancies.

While cutting off long leads after soldering, I made the amazing discovery that if I held onto them as I snipped, they didn't fly all over the room. It wouldn't work with the short leads. Still, one does learn.

The augat pins came all attached to a bar. Not knowing what augat pins were used for, I deduced from their shape that their function was to connect with something. What that something was I didn't know. Nor how. That left me with the problem of determining where to break them off. Since the instructions simply said to remove them, I wriggled one and did. The second one came off the same way. Hopefully it was the right way. A tiny pin remained inside. Filing that note to myself in the back of my mind, I proceeded with my soldering.

The manual suggested standing the board on edge between two objects to help with the soldering at this stage.

If all those holes were going to be filled, why put them there in the first place?

But I found my rubber-handled screwdriver was just the right size for propping up the board.

The next three augat pins were to be soldered in places marked P, CLR, and U. These pins came off the carrier even more readily than the first two. I pulled rather than wriggled. It was then I discovered that the central pinlet remained on the carrier. The augat pins were actually diminutive sockets—except for the first two, which were now effectively clogged. Off to borrow Susan's tweezers—nothing I had was fine enough to grasp the interior pins. Susan gave me a rather odd look when I inquired as to the whereabouts of her eyebrow tweezers in the middle of the night—but they worked.

Who's being disabled, the phantom, the memory, or me?

Soldering the three pins in place reminded me of the story about Nicholas Tesla's mother, who in her sixties was still able to pluck an eyelash and knot it twice. Some people obviously have more manual dexterity than others.

Two more augat pins were to go in mounting holes W and 0. A problem, however, revolved around the last pin. Seems you could put it in either hole 1 or hole 2. The option depended on some choices I couldn't make because I didn't understand them—and wouldn't until I had the computer itself. Back into the bag went the pin. When I had built the Sol itself, I'd tackle that problem.

Next came the DIP switch. The board had holes to accommodate fourteen pins. My switch had only twelve. The instructions assured me only twelve pins were used—and by now I'd come to accept little things like extra holes.

As a matter of fact the next step was to fill a lot of unwanted holes. If they were going to be filled, why put them there in the first place? Well, fill we must, as Con Edison used to say down in New York City.

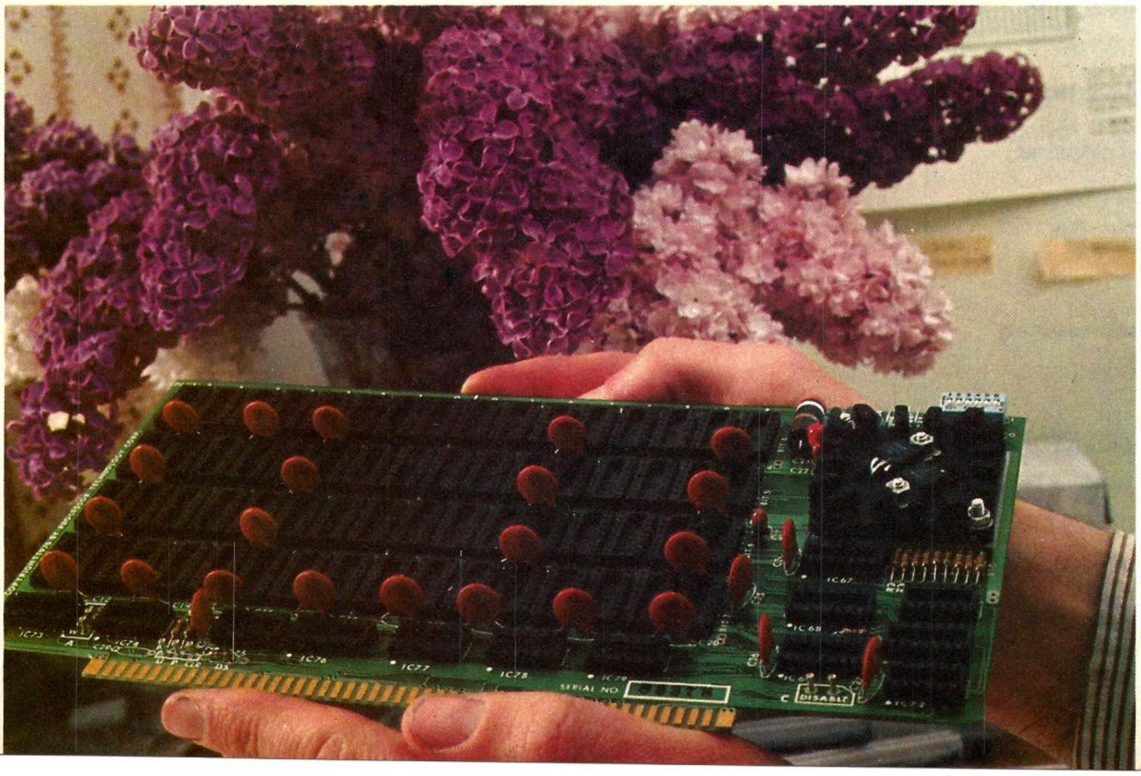
Jumpers are used to interconnect several points on the board. Once more I made the decision to wait, since which points you connected was dependent on which options you chose and I still didn't understand what those optional extras would do for me. A second rear-view mirror on the passenger side or a handy dandy inside-the-trunk light—such things any car salesman could convince me I didn't need. But a phantom memory disable? Do I want one? Who's being disabled, the phantom, the memory, or me?

So it was onwards to the home stretch, the IC sockets themselves. The catch was, they were to be located "with their end notches oriented as shown on the assembly drawing." Unfortunately, the Texas Instrument sockets that came with the kit didn't have any end notches. However, the depressed centers had little corners cut off on one side, just like IBM cards. Viewing this feature as a substitute for the notches, I soldered on those seventy-seven blankety-blank little sockets with fourteen to sixteen pins each. Which is a lot of soldering under the bridge.

The only thing that remained to be done was to place the actual chips in their sockets. Considering their sensitivity, I opted again to wait. I'd plug them in when I was ready to test the board. Meanwhile, elapsed time on the assembly project was twelve hours, mostly at one sitting. Apart from the saddle sores, I'd had a great time. I was ready for the Sol itself. After all, I couldn't test the absent-minded memory.

As if on cue, the Sol arrived the next day. The first thing I noticed on opening the package was a *Sol Systems Manual* thick enough to contain the construction directions for an ICBM.

Susan fished out the fabled walnut sides with the words "I'll just take these and lemon oil them," while I continued to stare at the manual. It was padded on top as if it were something one could safely beat one's head against during assembly. ▼



All right, here's the memory. Now where's the computer?

RAINBOW COMPUTING, INC.

Supplier of

WAVE MATE
THE DIGITAL GROUP
DEC PDP
Computer products

Peripherals and Supplies from
PERSCI
CENTRONIX
DIABLO
MAXELL

COMPUTER DEVICES
LEAR-SIEGLER
MULTI-TECH
TEXAS INSTRUMENTS

Specialists in Design, Implementation
and Support of
Custom Hardware/Software Systems for
Business, Educational, and Personal Use

Experts in most major computer
software including
CDC, IBM, PDP
BASIC, COBOL, FORTRAN, PL1
LISP, SIMULA, SNOBOL, SPSS, BMD's
COMPASS, MACRO,
6800, & Z80 assembly languages

10723 White Oak Ave., Granada Hills,
Ca. 91344
(213) 360-2171

THE COMPUTER STORE

Featuring

MITS Altair
Data General Micro Nova

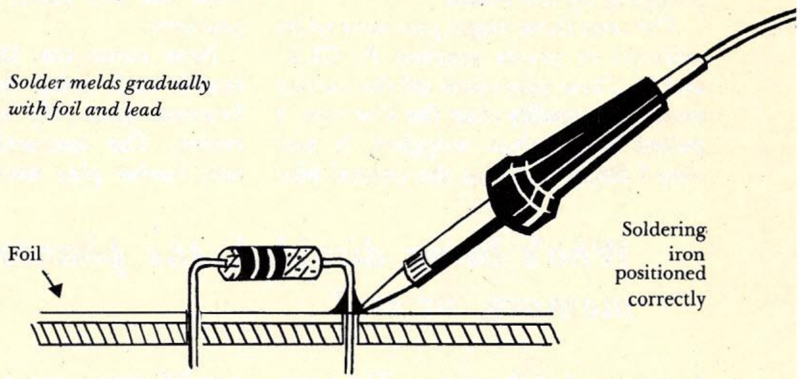
Also your headquarters for

Proto Typing Equipment
Wire Wrapping
Magnetic Supplies

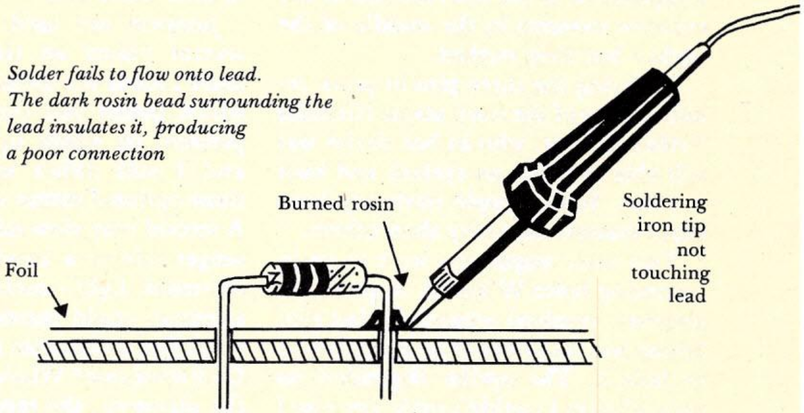
"Where personal computing begins."

THE COMPUTER STORE
63 SOUTH MAIN ST.
WINDSOR LOCKS,
CONN. 06096
(203) 627-0188

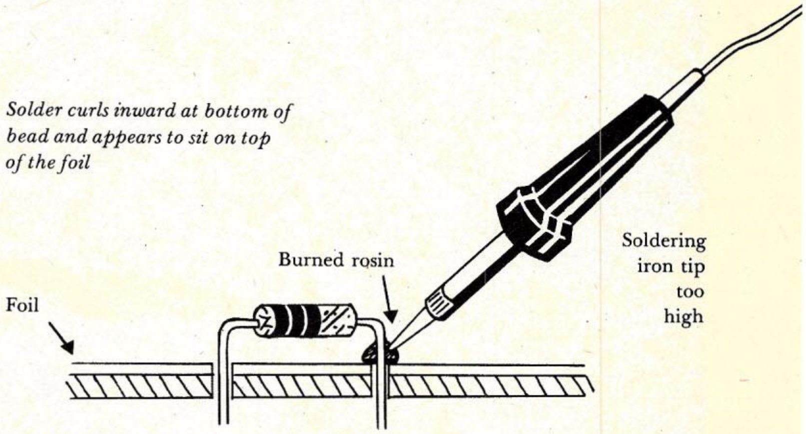
#1 GOOD SOLDERING CONNECTION


Solder melds gradually with foil and lead
Foil
Soldering iron positioned correctly

#2 POOR SOLDERING CONNECTION


Solder fails to flow onto lead. The dark rosin bead surrounding the lead insulates it, producing a poor connection
Burned rosin
Foil
Soldering iron tip not touching lead

#3 UNACCEPTABLE SOLDERING CONNECTION


Solder curls inward at bottom of bead and appears to sit on top of the foil
Burned rosin
Foil
Soldering iron tip too high

Illustrated by Joanne Zeiger

TOOLING UP

by Frank Becker

When you first get your kit, check or catalog all the parts. Most companies send a packing list with the kit and they instruct you to check all the packing bags to make sure you have all the parts. Not only do you thereby recheck the packers, you also find out which parts you have as extras. In one of my kits I didn't check all the parts, and when I had finished all the circuit boards I found two or three resistors and a couple of diodes left over. You can imagine the fun it is trying to check all the parts too late, after they are already on the board.

A cardboard box, cut in half, serves as a good stand for most of the parts, such as resistors, diodes, transistors, and IC chips. Because of the corrugated slots in the cardboard, it is easy to separate the different parts, and also it allows a place to write the different powers of the components. Once you get going on a kit it is a very helpful thing to have components separated by their powers because it saves time and confusion. You will find that a 5.05 resistor starts looking very much like a 50.5 resistor if you don't have them separated.

The tools you use also make the kit a lot easier to put together. The most important one of these is your soldering iron. You need a sharp tip because, as you get into the kit, you will find the soldering lugs seem to get smaller and smaller. On a typical circuit board you may find three or four IC chips that have six or eight pegs in a length of only ¾ of an inch. So it is important to have an iron that can get into these small places without hitting any other soldering lugs.

Needle-nosed pliers are also an essential part of your tool kit. Pliers are needed when putting together components such as wafer switches where many leads have to be soldered to the same lug. The wires from the leads may have to be twisted together to get a good solder joint. This can be quite difficult because of their small size and the small amount of room in which you work. Pliers will help you pull the wires through the small lugs, and they can also be helpful when soldering.

You also need a special area to work in. If you have a desk or table on which you can spread all the parts, it's a lot easier. Then, when you get a system going so you know where the wires are and where the parts are, the whole project will move faster and more easily. It's better to have a place that's out of everybody's way, because a small component may look like a small piece of scrap to a family-member in a clean-up mood. Believe me, it happens.

The day-to-day maintenance of your soldering iron deserves a lot of attention. It's important to keep the tip clean. It's also good policy to have a sponge or clean cloth on hand to wipe it off every so often. After you wipe it off, it is advisable to apply a small amount of solder to the tip. This is called tinning, and will protect the tip and enable you to make a good connection. Sometimes, if your iron is dirty, your solder will not stick or it will ball up around the solder peg. If you retin the iron it will help with the soldering.

A solder bridge is also something to watch out for when you start a kit. A solder bridge occurs when you touch another foil on the circuit board. It can happen if you use too much solder or if you drag the iron away from the joint you are working on. The foils are very small; a good rule, I have found, is to take a good look at the foil before you begin to work on it. Many times I've found that once you have put the heat to the joint, you can't really tell if you have really bridged the foil or not. A bridge can make a whole kit malfunction, but it is not hard to fix if you know that it's there.

If a soldering bridge does occur you can remove it by first heating the joint and then turning the circuit board upside down and shaking it lightly. If the solder doesn't fall off the joint, you may have to get a desoldering bulb. This tool is used with the soldering iron. Take the iron and heat the joint. Then, using the bulb, you can suck the unwanted solder off the joint. The foil side of the circuit board is usually covered with solder-resistant coating that helps prevent solder bridges. ▼

FREE

NCE Spring Catalog of Super Bargains!

Hard to Find Parts!
FOR COMPUTER EXPERIMENTERS

Many Items Like:
Numeric Printers
Micro® Key Switches
Burroughs Alpha-Numeric Displays
DC Power Supplies
Mini-Micro Computers
Peripherals and MORE!!

Name__________
Company__________
Address__________
City_____State_____Zip_____

NEWMAN COMPUTER EXCHANGE
1250 N. Main St. Dept. 65
Ann Arbor, Mich. 48104
Phone: 313/994-3200

Reconditioned Teletypes

Guaranteed 90-days on return basis

ASCR 33 $950 KSR 33 $595

Free delivery within 25 mile radius of New York City. Shipping extra beyond radius.

IMSAI edge connector and guides: ten for $39, $4.50 each.

COMPUTER MART OF NEW YORK
118 MADISON AVE
NEW YORK, NEW YORK 10016
(212) 686-7923

Deal Yourself in™...

A
K
Q
J
10
PC
77
A

Atlantic City, N.J.
August 27th-28th

What its all about!

Software Development
Micro Computers
Hardware Development
Disc Memories
Computer Comparisons
Interfacing
Program Implementation
AMSAT
Computerized Music
Video Terminals
Kit Construction
Printers
Computer Games
Digital Tapes

- Seminars and Technical talks by leading electronic equipment manufacturers
- Major Exhibits from all over the country
- Demonstrations in many areas including Home and Personal Computing
- Door Prizes, Free Literature and Free Mementos
- All this plus Sun and Surf - Fun and Excitement - Relaxation and Leisure

Personal™
Computing
77 Consumer Trade Fair

TICKETS FOR THE WEEKEND: INCLUDES EXHIBITS AND SEMINARS ON AUGUST 27-28th : $10.00 at the door; or ORDER BEFORE AUGUST 10th and SAVE 20%. and NO WAITING IN LINE!

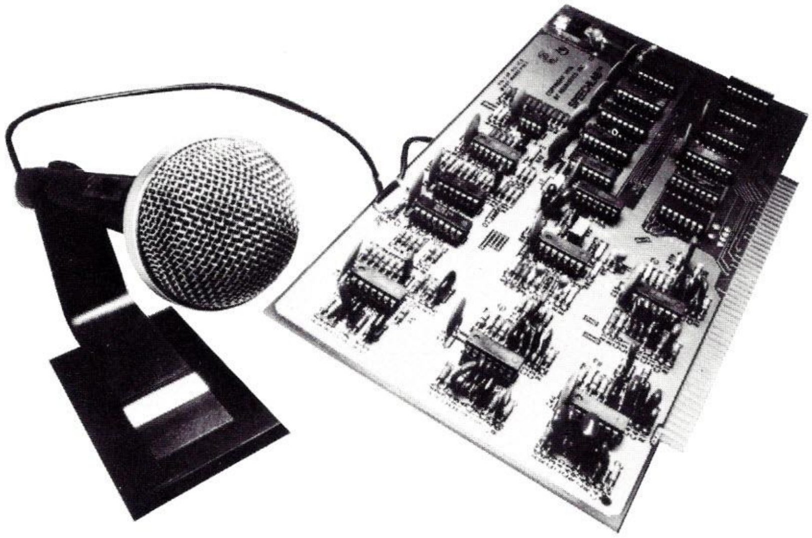
SEND CHECK TO "PERSONAL COMPUTING 77"
Rt 1, Box 242, Mays Landing, N.J. 08330

c

TALK TO ME

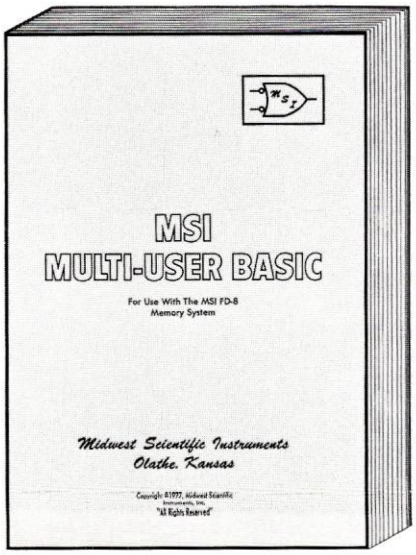
Speak and your S-100 bus computer will listen—if you have a Heuristics SpeechLab. Available in kit form for $249 or preassembled and tested for $299, the SpeechLab digitizes data into a digestible form for your Sol, Imsai, or Altair. Other computers can be eared in with the aid of a separate connector and power supply. Price includes a complete hardware and BASIC software system, not to mention a thick laboratory manual that will keep any curious computerist turning the pages.

The SpeechLab has sixty-four bytes of storage per spoken word and a vocabulary of up to sixty-four words in memory with a ninety-five percent recognition rate. Open sesame.



GROUPIE BASIC—6800

Looking for Multi-User BASIC with program and Data File handling capability for your 6800? Well, Midwest Scientific has it. For use with the FD-8, the system is set up to be accessed by up to four users simultaneously with minimal effect on execution speed. With or without disk memory, it looks like a plum for the 6800 fans.


MSI
MSI
MULTI-USER BASIC
For Use With The MSI FD-8 Memory System
Midwest Scientific Instruments
Olathe, Kansas
Copyright ©1977, Midwest Scientific Instruments, Inc.
"All Rights Reserved"

CATTY CHATTER

commodore
pet computer 2001

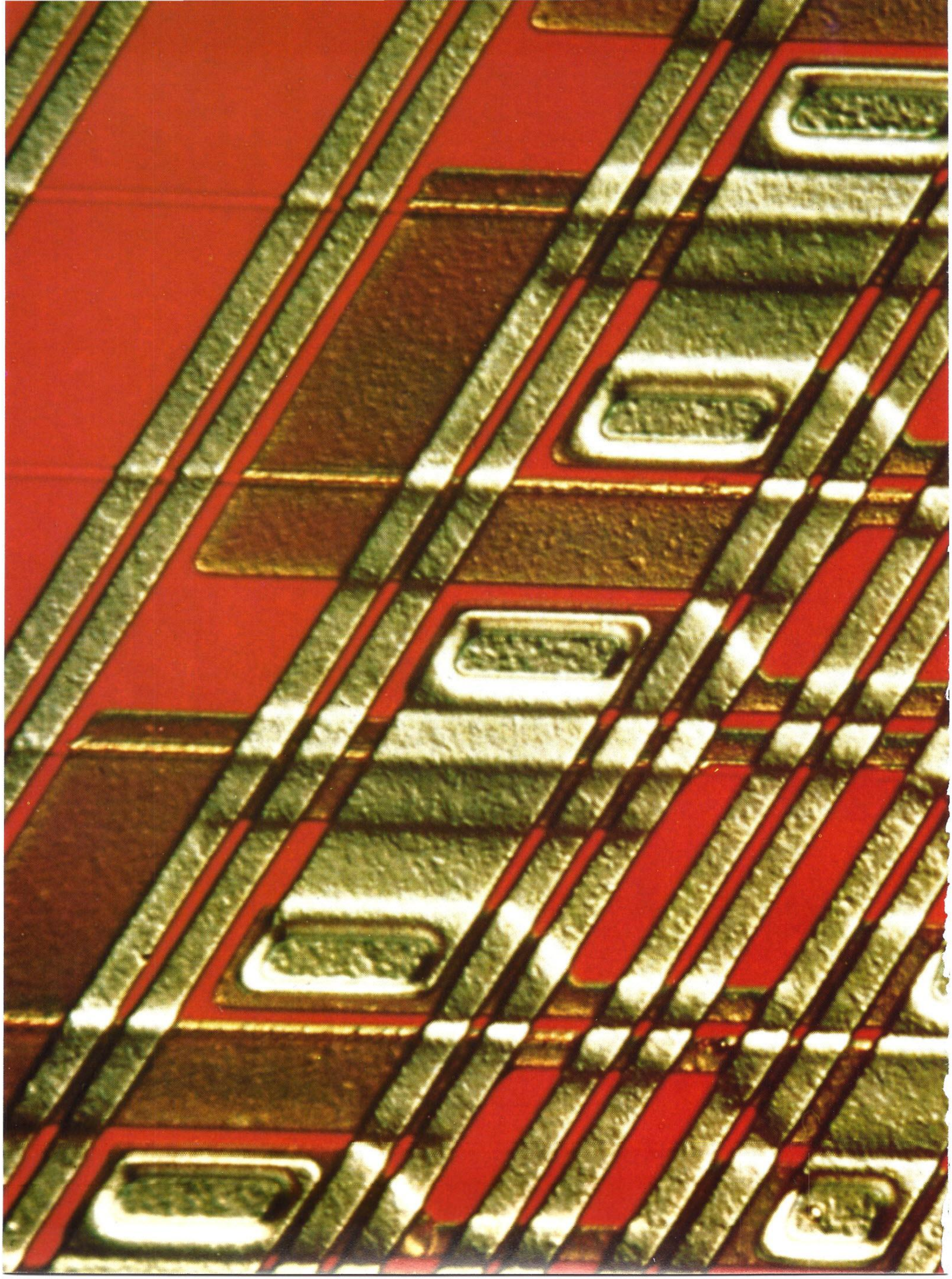
The press has been feeding on Pet (Personal Electronic Transactor) stories for half a year now. Well, the 2001 is up and running and should be on sale by the time you read this. With a 4K (expandable to 32K) RAM and 8K BASIC interpreter, 4K operating system ROM, a modernistic module topped off by a nine-inch high-resolution CRT, the Commodore entry into the personal computing field looks as if it will have muscle—If (and it's a big if) Commodore markets good software as well as they do calculators.

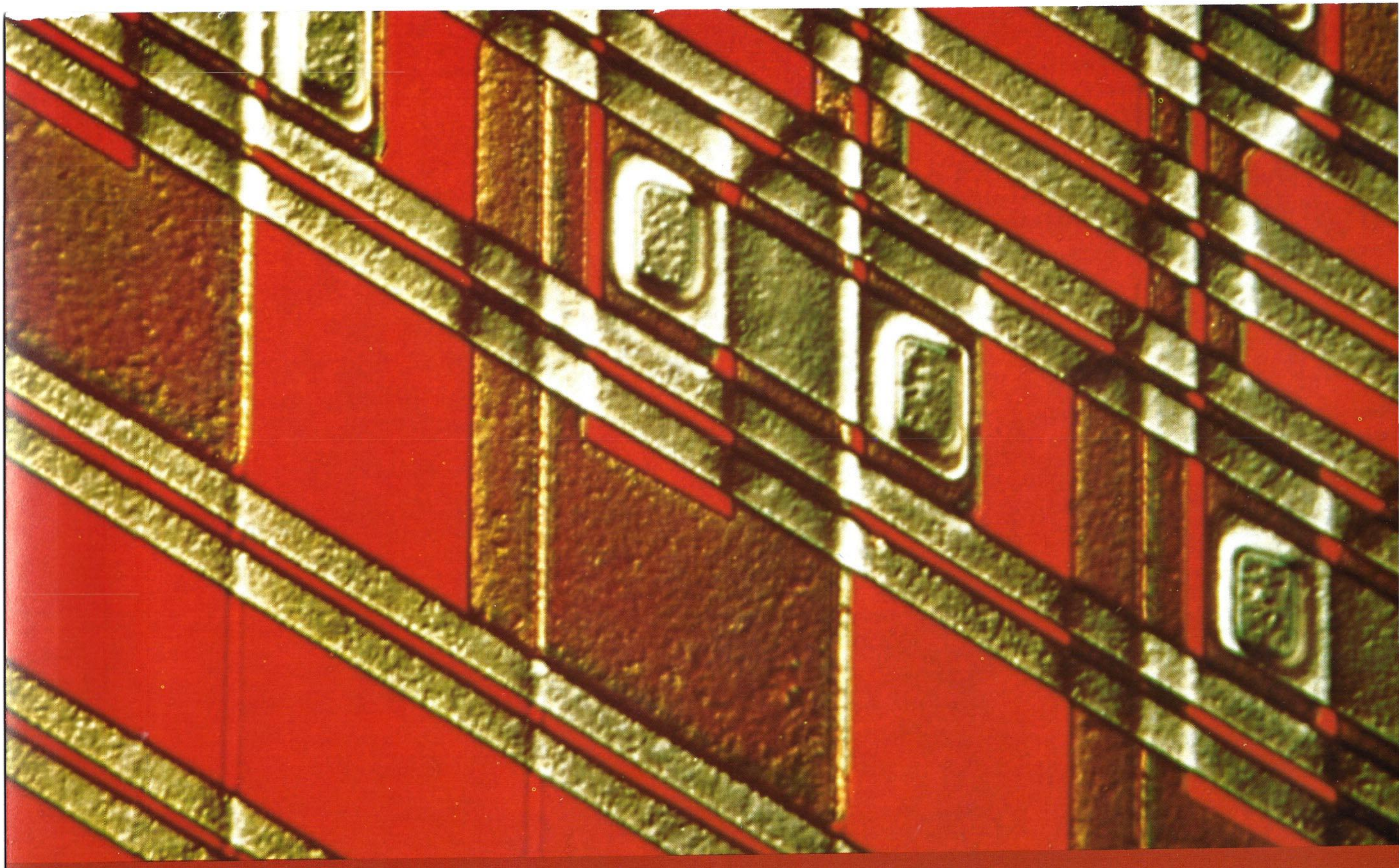
BUILD YOUR OWN FLOPPY

SWTPc
MINIFLOPPY DISK SYSTEM

Memory prices keep tumbling as more and more manufacturers get on the storage train. The latest entrant is Southwest Technical Products Corp. with the MF-68 Minifloppy Kit for $995. Designed as a partner for SWTP's 6800, the MF-68 comes with both DISK BASIC and a floppy disk operating system. Commands include CREATE, SAVE, RUN, LOAD, PURGE, PACK, CATALOG, RENAME, INITIALIZE, and PATCH.

As your system grows, the Minifloppy can be expanded to its four-drive limit with an $850 MF-6X expansion kit. The drive used in both kits is ye old reliable Shugart.





August 1977

ROM

COMPUTER APPLICATIONS FOR LIVING

One approach to very-high-density computer storage of the future is this experimental IBM magnetic "bubble" lattice device (here, bubbles are invisible). This .00007 sq. in. section can hold 350 bubbles—or bits of information—a density of 5 million bits per sq. in.

Courtesy of IBM

Sol Terminal Computer
Processor Technology

One Sol-20 equals three computers.

To do real work with any computer, big or small, it takes a complete system. That's one of the nice things about the Sol-20. It was built from the ground-up as the heart of three fixed price computer systems with all the peripheral gear and software included to get you up and on the air.

Sol System I costs just $1649 in kit form or $2129 fully burned in and tested. Here's what you get: a Sol-20 with the SOLOS personality module for stand alone computer power, an 8192 word memory, a 12″ TV/video monitor, a cassette recorder with BASIC software tape and all necessary cables.

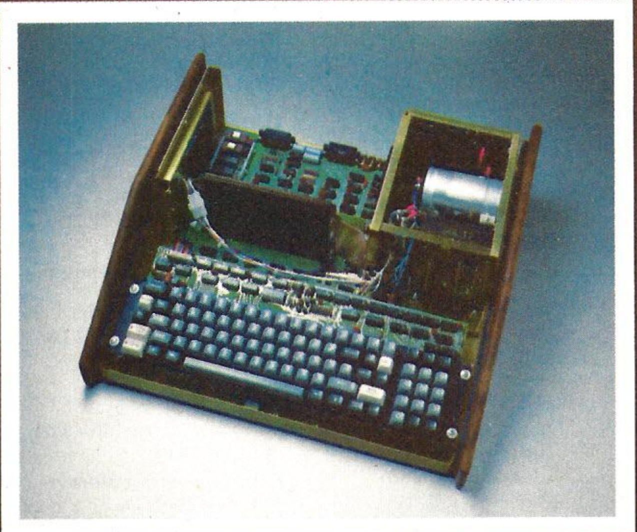
Sol System II has the same equipment plus a larger

Helios II
Processor

capacity 16,384 word memory. It sells for $1883 in kit form; $2283 fully assembled.

For even more demanding tasks, Sol System III features Sol-20/SOLOS, a 32,768 word memory, the video monitor, Helios II Disk Memory System and DISK BASIC Diskette. Price, $4750 in kit form, $5450 fully assembled and tested.

And remember, though we call these small or personal computer systems, they have more power per dollar than anything ever offered. They provide performance comparable with mini-computer systems priced thousands of dollars more.



The functional beauty of Sol Computer Systems is more than skin deep. A look inside reveals a simple elegance of design and sturdy construction.

The Small Computer Catalog for the rest of the real computer system story.

Visit your local computer store for a copy of our fully illustrated 22 page catalog. Or you may write or call us if more convenient. Please address Processor Technology, Box O, 6200 Hollis Street, Emeryville, CA 94608. (415) 652-8080.

Processor Technology
Corporation

See Sol here...

ALABAMA

ICP, Computerland
1550 Montgomery Hwy
Birmingham, AL 35226
(205) 979-0707

ARIZONA

Byte Shop Tempe
813 N. Scottsdale Rd.
Tempe, AZ 85281
(602) 894-1129

Byte Shop Phoenix
12654 N. 28th Dr.
Phoenix, AZ 85029
(602) 942-7300

Byte Shop Tucson
2612 E. Broadway
Tucson, AZ 85716
(602) 327-4579

CALIFORNIA

The Byte Shop
1514 University Ave.
Berkeley, CA 94703
(415) 845-6366

Byte Shop Computer Store
6041 Greenback Lane
Citrus Heights, CA 95610
(916) 961-2983

Computer Center
1913 Harbor Blvd.
Costa Mesa, CA 92627
(714) 646-0221

Data Consultants, Inc.
2350 W. Shaw, Suite 114
Fresno, CA 93711
(209) 431-6461

Bits 'N Bytes
679 S. State College Blvd.
Fullerton, CA 92631
(714) 879-8386

The Byte Shop
16508 Hawthorne Blvd.
Lawndale, CA 90260
(213) 371-2421

The Byte Shop
1063 El Camino Real
Mountain View, CA 94040
(415) 969-5464

Digital Deli
80 W. El Camino Real
Mountain View, CA 94040
(415) 961-2828

The Computer Mart
624 West Katella #10
Orange, CA 92667
(714) 633-1222

The Byte Shop
2227 El Camino Real
Palo Alto, CA 94306
(415) 327-8080

Byte Shop
496 South Lake Ave.
Pasadena, CA 91101
(213) 684-3311

The Computer Store
of San Francisco
1093 Mission Street
San Francisco, CA 94103
(415) 431-0640

Byte Shop
321 Pacific Ave.
San Francisco, CA 94111
(415) 421-8686

The Byte Shop
2626 Union Avenue
San Jose, CA 95124
(408) 377-4685

The Computer Room
124H Blossom Hill Rd.
San Jose, CA 95123
(408) 226-8383

The Byte Shop
509 Francisco Blvd.
San Rafael, CA 94901
(415) 457-9311

The Byte Shop
3400 El Camino Real
Santa Clara, CA 95051
(408) 249-4221

Recreational Computer
Centers
1324 South Mary Ave.
Sunnyvale, CA 94087
(408) 735-7480

Byte Shop of Tarzana
18424 Ventura Blvd.
Tarzana, CA 91356
(213) 343-3919

The Byte Shop
2989 North Main St.
Walnut Creek, CA 94596
(415) 933-6252

Byte Shop
14300 Beach Blvd.
Westminster, CA 92683
(714) 894-9131

COLORADO

Byte Shop
2040 30th St.
Boulder, CO 80301
(303) 449-6233

FLORIDA

Sunny Computer Stores
University Shopping
Center
1238A S. Dixie Hwy.
Coral Gables, FL 33146
(305) 661-6042

Delta Electronics
2000 U.S. Hwy. 441 East
Leesburg, FL 32748
(904) 357-4244

Byte Shop of Miami
7825 Bird Road
Miami, FL 33155
(303) 264-2983

Microcomputer
Systems Inc.
144 So. Dale Mabry Hwy.
Tampa, FL 33609
(813) 879-4301

GEORGIA

Atlanta Computer Mart
5091-B Buford Hwy.
Atlanta, GA 30340
(404) 455-0647

ILLINOIS

The Numbers Racket
623½ South Wright St.
Champaign, IL 61820
(217) 352-5435

itty bitty machine co.
1316 Chicago Ave.
Evanston, IL 60201
(312) 328-6800

Reeves Communications
1550 W. Court St.
Kankakee, IL 60901
(815) 937-4516

itty bitty machine co.
42 West Roosevelt
Lombard, IL 60148
(312) 620-5808

INDIANA

The Data Domain
406 So. College Ave.
Bloomington, IN 47401
(812) 334-3607

The Byte Shop
5947 East 82nd St.
Indianapolis, IN 46250
(317) 842-2983

The Data Domain
7027 N. Michigan Rd.
Indianapolis, IN 46268
(317) 251-3139

The Data Domain
219 West Columbia
West Lafayette, IN 47905
(317) 743-3951

KENTUCKY

The Data Domain
3028 Hunsinger Lane
Louisville, KY 40220
(502) 456-5242

MICHIGAN

The Computer Store
of Ann Arbor
310 East Washington
Ann Arbor, MI 48104
(313) 995-7616

Computer Mart
or Royal Oak
1800 W. 14 Mile Rd.
Royal Oak, MI 48073
(313) 576-0900

Genral Computer Store
2011 Livernois
Troy, MI 48084
(313) 362-0022

NEW JERSEY

Hoboken Computer Works
No. 20 Hudson Place
Hoboken, NJ 07030
(201) 420-1644

The Computer Mart
of New Jersey
501 Route 27
Iselin, NJ 08830
(201) 283-0600

NEW YORK

The Computer Mart
of Long Island
2072 Front Street
East Meadow, L.I.,
NY 11554
(516) 794-0510

Synchro Sound
Enterprises
193-25 Jamaica Ave.
Hollis, NY 11423
(212) 359-1489

The Computer Shoppe
444 Middle Country Rd.
Middle Island, NY 11953
(516) 732-3086

Audio Design Electronics
487 Broadway, Ste. 512
New York, NY 10013
(212) 226-2038

The Computer Mart
of New York
118 Madison Ave.
New York, NY 10001
(212) 686-7923

The Computer Corner
200 Hamilton Ave.
White Plains, NY 10601
(914) 949-3282

OHIO

Cybershop
1451 S. Hamilton Rd.
Columbus, OH 43227
(614) 239-8081

OKLAHOMA

High Technology
1020 West Wilshire Blvd.
Oklahoma City, OK 73116
(405) 842-2021

OREGON

Byte Shop Computer Store
3482 S.W.
Cedar Hills Blvd.
Beaverton, OR 97005
(503) 644-2686

The Real Oregon
Computer Co.
205 West 10th Ave.
Eugene, OR 97401
(503) 484-1040

Byte Shop Computer Store
2033 S.W. 4th Ave.
Portland, OR 97201
(503) 223-3496

RHODE ISLAND

Computer Power, Inc.
M24 Airport Mall
1800 Post Rd.
Warwick, RI 02886
(401) 738-4477

SOUTH CAROLINA

Byte Shop
2018 Green Street
Columbia, SC 29205
(803) 771-7824

TENNESSEE

Microproducts & Systems
2307 E. Center St.
Kingsport, TN 37664
(615) 245-8081

TEXAS

Byte Shop
3211 Fondren
Houston, TX 77063
(713) 977-0664

Computertex
2300 Richmond Ave.
Houston, TX 77098
(713) 526-3456

Interactive Computers
7646½ Dashwood Rd.
Houston, TX 77036
(713) 772-5257

The Micro Store
634 So. Central
Expressway
Richardson, TX 75080
(214) 231-1096

VIRGINIA

The Computer Systems
Store
1984 Chain Bridge Rd.
McLean, VA 22101
(301) 460-3634

Media Reactions Inc.
11303 South Shore Dr.
Reston, VA 22090
(703) 471-9330

WASHINGTON

Byte Shop Computer Store
14701 N.E. 20th Ave.
Bellevue, WA 98007
(206) 746-0651

The Retail Computer Store
410 N.E. 72nd
Seattle, WA 98115
(206) 524-4101

WISCONSIN

The Milwaukee
Computer Store
6916 W. North Ave.
Milwaukee, WI 53213
(414) 259-9140

CANADA

Trintronics
160 Elgin St.
Place Bell Canada
Ottawa, Ontario K2P 2C4
(613) 236-7767

First Canadian Computer
Store, Ltd.
44 Eglinton Ave. West
Toronto, Ontario M4R 1A1
(416) 482-8080

The Computer Place
186 Queen St. West
Toronto, Ontario M5V 1Z1
(416) 598-0262

Pacific Computer Store
4509-11 Rupert St.
Vancouver, B.C. V5R 2J4
(604) 438-3282

Processor
Technology
Corporation

Binary Clocks

by
Caxton
C.
Foster

Most clocks first divide time into discrete intervals and then count and display, by the angle of rotation of one or more shafts, the number of intervals that have passed. That is to say, the pendulum, or quartz crystal, or balance wheel are digital in nature, quantizing the flow of time into discrete events (ticks). These quanta of time are summed by the rotation of the escape wheel and, by suitable gearing, are displayed by the hands (which indicate the time, in analog fashion, on the familiar clock face). By close examination of a large clock dial (such as a grandfather's with a long period pendulum) one can see the very small jumps made by the minute hand as the seconds tick by, but for most clocks the indication appears to be continuously variable.

> *My nine-(1001)-year-old can tell time better with this clock than with a conventional one (0001).*

The so-called *digital* or *ticket* clock works in just the opposite fashion. A smooth shaft rotation is converted to the flipping of tickets which indicate the time in discrete, integer minutes and hours.

The two clocks described here are both digital clocks but the number system they use is binary rather than decimal. In addition to using the radix two preferred by most computers, these clocks use a mechanical form of computer type logic in their functioning.

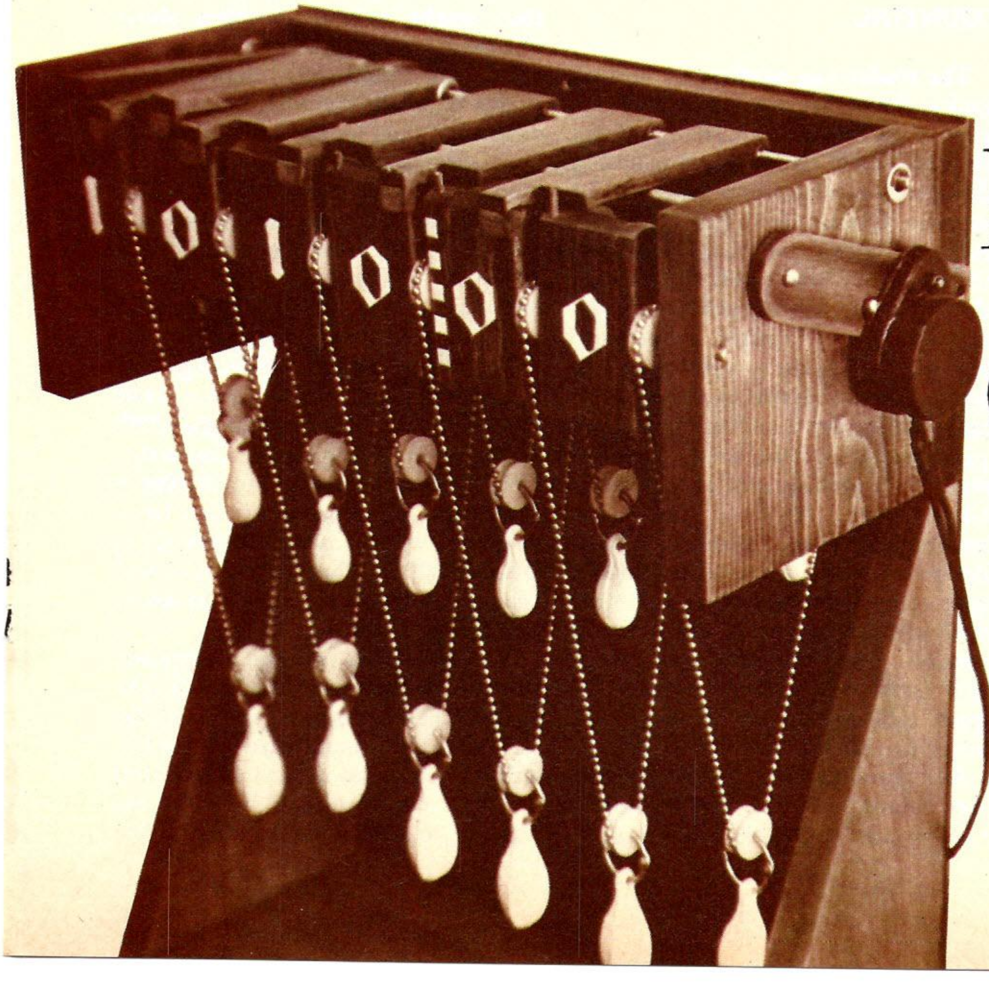
MODEL TWO

We will describe the second clock first because it is somewhat simpler to explain. First we must discuss the binary number system. Only the numbers *one* and *zero* are used. As in the decimal (common) number system, position is important. For example, in the decimal system 1, 10, 100, and 1000 are all different whereas 10, 010, and 0010 are the same because leading zeroes are ignored. In the binary number system the numbers 1, 10, 100, and 1000 represent the quantities whose names (in English) are one, two, four, and eight. Adding another trailing zero multiplies a number, not by 10 as in the decimal system, but by two.

The numbers from zero through twelve in both decimal and binary are:

0 = 0000
1 = 0001
2 = 0010
3 = 0011
4 = 0100
5 = 0101
6 = 0110
7 = 0111
8 = 1000
9 = 1001
10 = 1010
11 = 1011
12 = 1100

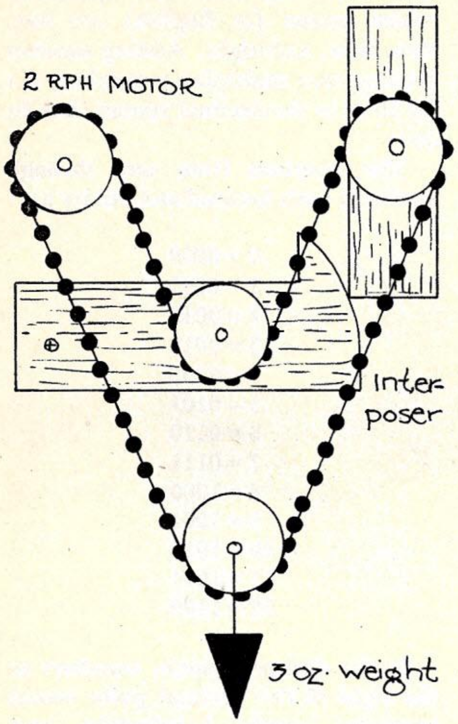
In the decimal system, numbers to the right of the decimal point would represent tenths, hundredths, and thousandths of an hour—although we don't usually express time this way. In the binary system, numbers to the right of the *binary point* represent



Copyright © 1976 by *The Bulletin of the National Association of Watch and Clock Collectors.* Reprinted by permission of the publisher.

Photographs by Andrew Singer

Figure 1. Basic timing mechanism.



halves, quarters, eighths, sixteenths, thirty-seconds, and sixty-fourths of an hour.

In Clock Two only the quarter hours are indicated, thus:

quarter past = 01
half past = 10
quarter of = 11
full hour = 00

or one, two, three, and no quarters.

For example then:

0101 01

would be quarter past five. With practice, it all gets simpler and my nine-(1001)-year-old can tell time better with this clock than with a conventional one (0001).

There are three (0011) key parts to a binary clock such as this one: the timing device, the counting mechanism, and the noon reset. We will discuss them in turn.

TIMING

Figure 1 shows the basic timing mechanism. As the two-revolution-per-hour motor turns, it pulls up the three-ounce weight and lets the interposer drop down. When enough bead chain has been fed through, the tip of the interposer drops below the bottom of the tumbler and, with nothing to stop it, the tumbler now is pulled clockwise by the three-ounce weight. This, of course, pulls up the interposer and after half a revolution the tumbler comes to a stop against the interposer with a satisfying click.

Since the motor-driven sprocket makes one turn in half an hour it basically moves 24 beads from the lower loop to the upper every 30 minutes. In order that the lower loop not run out of beads the tumbler must return exactly 24 beads in the same time period. That represents one full or two half turns. Thus, the tumbler tumbles once every 15 minutes. On one face we have painted a ZERO. On the other, a ONE. From reading this face we can see whether an odd (ONE) or even (ZERO) number of quarter hours have elapsed. For estimates of time closer than the nearest 15 minutes, the front vertical face of the interposer has three marks on it to indicate five minute intervals.

Conversation slows down and the visitors steal ever more frequent glances at the clock.

COUNTING

The reader can confirm by looking back at the binary numbers above that when a particular digit changes from ONE to ZERO its left-hand neighbor changes from whatever it is now to the opposite. Changes from ZERO to ONE have no effect on their left-hand neighbors. To make this action take place in our clock, we need three elements: a device to make a free tumbler turn, a device to restrain a free tumbler, and a device to release a restrained tumbler. The first of these is a sprocket glued to the tumbler with a bead-chain passing over the sprocket. A heavy weight (three ounces) on the front end of the chain causes the tumbler to *want* to turn top forward. A light-weight (one ounce) on the other end keeps the chain well seated in the sprocket. The tumbler is restrained from rotation by a pawl resting on its upper right-hand side (as seen when facing the clock—see Figure 2).

As a tumbler turns half way round from showing ONE to the position in which it will display a ZERO, a short (¼ inch) pin on its end knocks up the pawl above it, releasing its left-hand neighbor. The pawl drops back down to catch the rotating tumbler by the time it has gone half a turn.

A note on construction technique: the pawls must be sanded away about 1/16-inch or so on their bottom side. This is so that the right-hand tumbler when going from ZERO to ONE won't *nudge* it. The pawl is held from dropping by the left-hand, at-this-point-stationary, tumbler and the cut away allows clearance for the right-hand tumbler to go by.

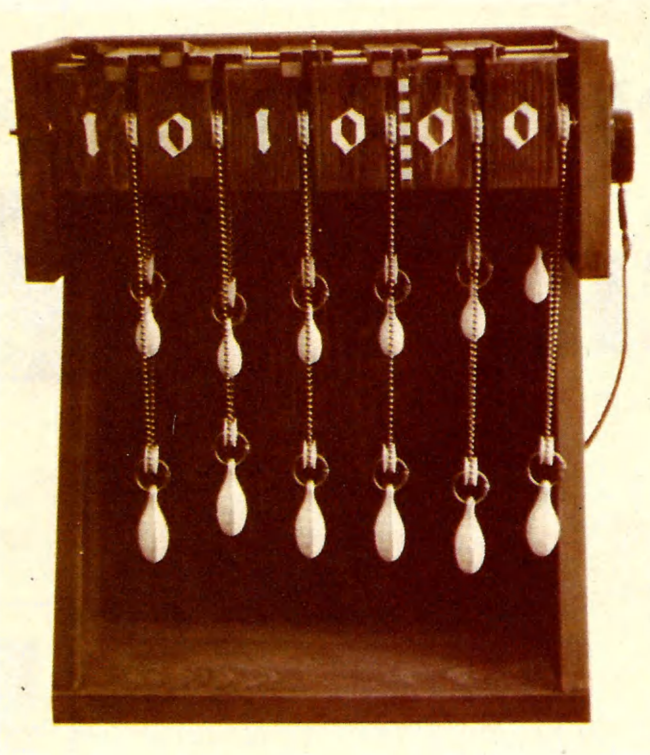
RESET

The remaining aspect of Clock Two that we must consider is the reset mechanism. Conventional clocks count hours as: ...., ten, eleven, twelve, one, two, ... Four binary digits would, if left to themselves, count up to fifteen, then go to zero and start over again. Logical as it might be to divide the day into sixteen or perhaps thirty-two units of time, we felt that there might be some problem about this catching on right away so we decided to retain the conventional twelve-hour scheme with one modification. That is, we will call the twelfth hour *the zero hour* and count: ...., nine, ten, eleven, zero, one, ... This sounds a bit strange at noon but is in perfect accord with the midnight to one A.M. designations on a twenty-four hour clock. (For example, 0015 hours is quarter past midnight.)

To accomplish this we arrange that when our clock tries to show 1100 (twelve) we do not restrain the ONE in the third position from the right. This ONE spins around to display a ZERO and gives a carry to its left-hand neighbor which then changes from ONE to ZERO. Thus the pattern 1011 (eleven) is followed, almost at once, by 0000 (zero hours).

To accomplish this the restraining pawl of the four-hour tumbler is split in half. The protrusion on the under side of the right-hand part lifts the left half whenever the right half is kicked up by a carry coming in from the tumbler to the right. As long as the

Figure 2.
Front view of
Model Two.



eight-hour tumbler shows ZERO, either both halves of the tumbler engage and restrain the four-hour tumbler (when it shows ZERO), or at least the left half does (when it shows ONE). Now, when the eight-hour tumbler shows a ONE the pin on its top lifts up the U-shaped left half of the four-hour pawl, leaving only the right half to restrain the tumbler. When the four-hour tumbler shows ZERO all is normal but when it flips over to show ONE the notch in its end allows it to slip right past the right half of the pawl. It therefore goes from ZERO through ONE and back to ZERO in one swift rotation. But in going from ONE to ZERO it sends a carry to the eight-hour tumbler which then goes from ONE to ZERO. Thus the clock goes from showing eleven (1011) through twelve (1100) to zero (0000).

This clock has been running satisfactorily for six months and keeps as good time as any other electric clock. The chain on the half-hour tumbler was doubled to reduce the frequency of winding. Even so, the half and one-hour weights must be raised every second day. This could be increased to once every four days by using twelve-hole sprockets instead of twenty-four-hole, but when I built the clock I used what I had on hand.

MODEL ONE

The original model was similar in its basic reset and display methods but differed in the number of tumblers and in the provision of an automatic rewind mechanism. Model One had an additional hour tumbler, making it display twenty-four hours and four additional fractional hour tumblers (eleven tumblers in all), allowing it to display sixty-fourths of an hour. Rather than trying to make a reset mechanism that would convert sixty-four into sixty by resetting to zero at sixty, I decided to allow the natural binary division of the hour into sixty-four parts or *binits* and accept sixteen binits per quarter hour.

The more interesting aspect of Model One was the rewind mechanism. Power is provided by a 100-ounce inch synchronous motor turning one revolution-per-minute. Nylon gears of thirty and sixteen teeth convert this to thirty-two revolutions per hour of a back shaft.

First there is a twenty-four-hole sprocket on the end of the back shaft that provides twelve beads every binit which are absorbed by a half turn of the one-binit tumbler.

There is a continuous loop of bead chain for each tumbler (except the one-binit tumbler), which goes over the sprocket to a three-ounce weight supported from a pulley, up to the back shaft where it makes four turns around the back shaft, down to a one-ounce weight supported from a pulley and hence to the sprocket once again.

As the back shaft turns, the friction of the four turns of bead chain raises the three-ounce weight and lowers the one-ounce weight until the one-ounce weight is at the end of its string. At this point the string is supporting part of the one-ounce weight and tension on the chain is reduced, thus reducing friction on the back shaft and allowing the chain to slip until the turning of the tumbler raises the one-ounce weight and the process starts again.

The weights must be chosen quite carefully to provide enough torque to turn the tumbler and still have a small enough difference to allow the friction with the soon-highly-polished back-shaft to lift the larger weight.

Another problem arises with this version of the clock. Since the chain is wound around the back-shaft, it tends to "walk" towards one end dragging the tumblers with it and eventually jamming the action because of pressure on the end-most tumbler. To counteract this, every second chain is wound around the shaft *backwards.* Thus each pair of tumblers press together but they don't all *gang up* on the end one.

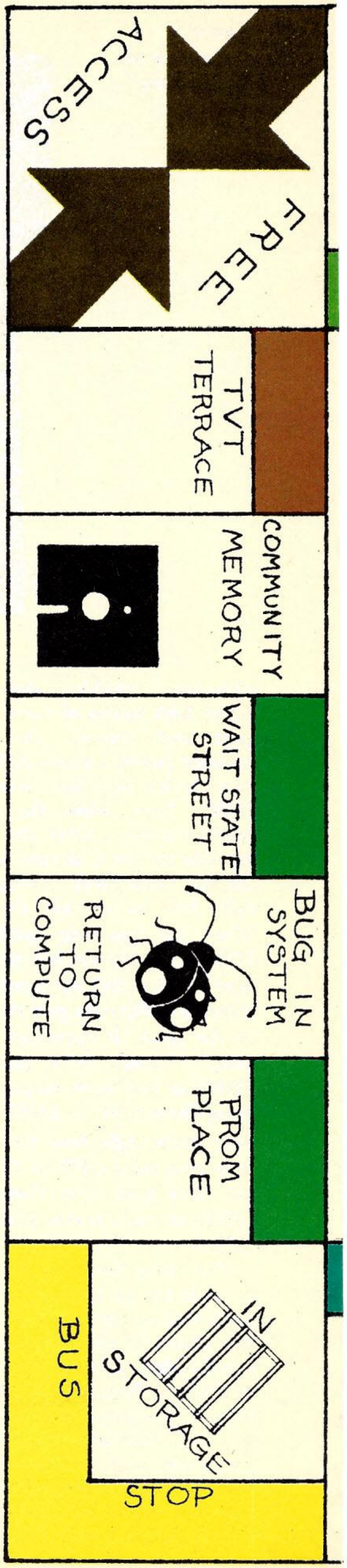
The final difficulty with Model One is that I have found it impossible to conduct a conversation in an office where it is running. It is not that the clacking of the tumblers every fifty-five seconds or so is loud. It is that when a *big change* is about to ensue, say from 0111 to 1000, conversation slows down and the visitors steal ever more frequent glances at the clock, until finally all pretense of paying attention to you is given up and they turn to watch the clock full-time. When the tumblers finally clatter over, the nervous tension is released and you have another binit to make your point before the visitor's attention begins to wander again. I don't know if it is a basic distrust of my engineering ability that causes them to wait to see if all works properly, or just a natural human desire for closure or completion of a cycle. Whatever the reason may be, I gave Model One to the computer science museum at the university where I teach. There people can and do spend minutes (excuse me, binits) on end staring at the *world's largest binary digital clock.* ▼

COMPUTER POWER

&

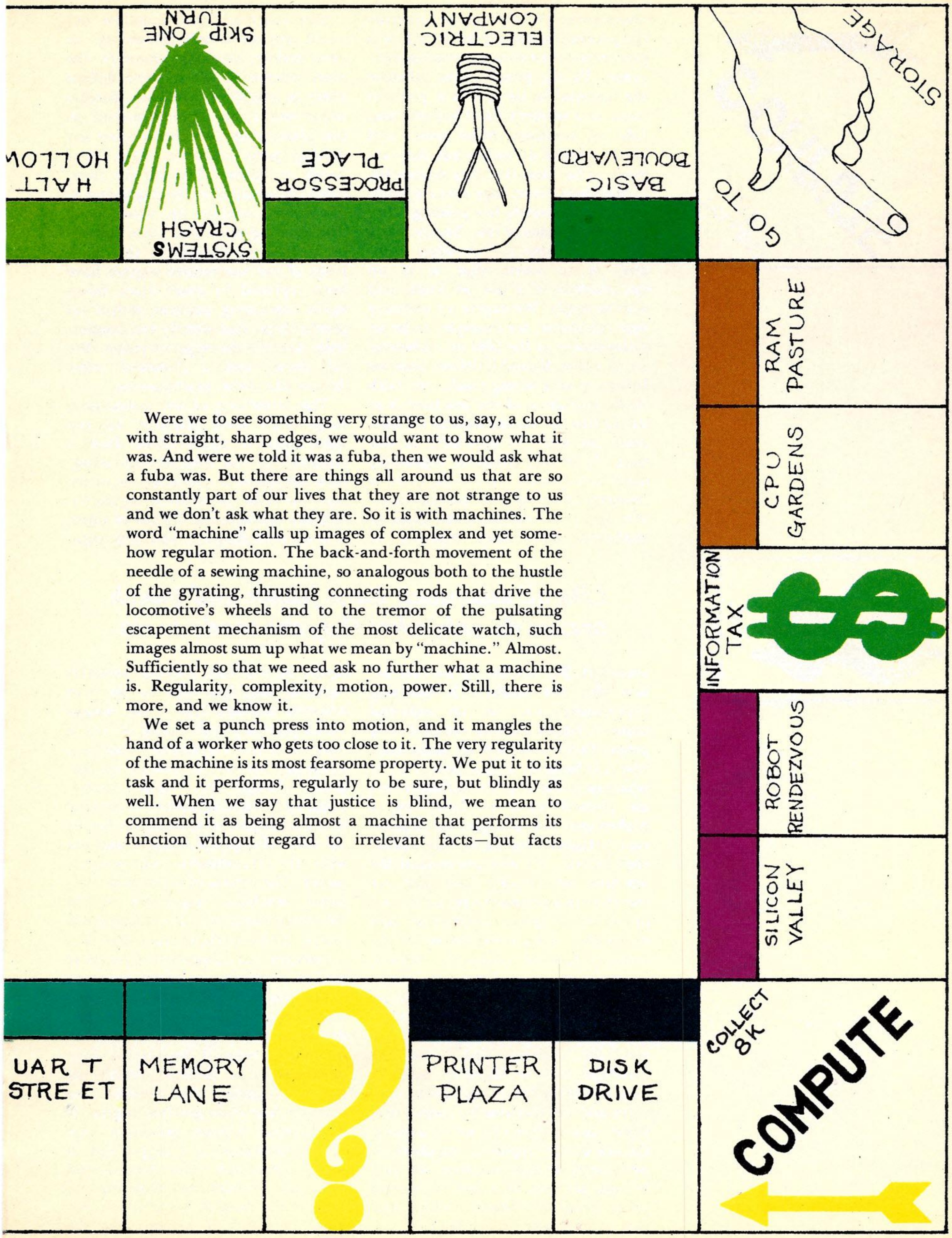
Where It Comes From

by Joseph Weizenbaum


FREE ACCESS
TVT TERRACE
COMMUNITY MEMORY
WAIT STATE STREET
BUG IN SYSTEM
RETURN TO COMPUTE
PROM PLACE
IN STORAGE
BUS
STOP

Illustrations by Cindy Hain

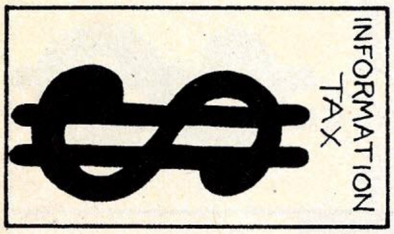
From *Computer Power and Human Reason,* Copyright © 1976 by W. H. Freeman and Co.


HALT HOLLOW
SYSTEMS CRASH
SKIP ONE TURN
PROCESSOR PLACE
ELECTRIC COMPANY
BASIC BOULEVARD
GO TO STORAGE
RAM PASTURE
CPU GARDENS
INFORMATION TAX
ROBOT RENDEZVOUS
SILICON VALLEY
COLLECT 8K
COMPUTE
DISK DRIVE
PRINTER PLAZA
MEMORY LANE
UART STREET

Were we to see something very strange to us, say, a cloud with straight, sharp edges, we would want to know what it was. And were we told it was a fuba, then we would ask what a fuba was. But there are things all around us that are so constantly part of our lives that they are not strange to us and we don't ask what they are. So it is with machines. The word "machine" calls up images of complex and yet somehow regular motion. The back-and-forth movement of the needle of a sewing machine, so analogous both to the hustle of the gyrating, thrusting connecting rods that drive the locomotive's wheels and to the tremor of the pulsating escapement mechanism of the most delicate watch, such images almost sum up what we mean by "machine." Almost. Sufficiently so that we need ask no further what a machine is. Regularity, complexity, motion, power. Still, there is more, and we know it.

We set a punch press into motion, and it mangles the hand of a worker who gets too close to it. The very regularity of the machine is its most fearsome property. We put it to its task and it performs, regularly to be sure, but blindly as well. When we say that justice is blind, we mean to commend it as being almost a machine that performs its function without regard to irrelevant facts—but facts

COMPUTE
COLLECT 8K



nonetheless. To blind justice, whether the prisoner before the bar is rich or is poor or is a man or is a woman is irrelevant. To the punch press, whether the material in its jaws is a piece of metal or a worker's hand is irrelevant. Like all machines, blind justice and punch presses do only what they are made to do—and that they do exactly.

Machines, when they operate properly, are not merely law abiding; they are embodiments of law. To say that a specific machine is "operating properly" is to assert that it is an embodiment of a law we know and wish to apply. We expect an ordinary desk calculator, for example, to be an embodiment of the laws of arithmetic we all know. Should it deliver what we believe to be a wrong result, our faith in the lawfulness of the machine is so strong that we usually assume we have made an error in punching in our data. It is only when it repeatedly malfunctions that we decide there is "something wrong with the machine." We never believe that the laws of arithmetic have been repealed or amended. But neither do we ever believe that the machine is behaving capriciously, i.e., in an unlawful manner. No, in order to restore it to its proper function we seek to understand why it behaves as it *now* does, i.e., of what law it is now an embodiment. We are pleased when we find, say, a broken gear that *accounts* for its aberrant behavior. We have then discovered its law. We now understand the machine we actually have and are therefore in a position to repair it, i.e., to convert it to the machine we had originally, to an embodiment of the ordinary laws of arithmetic. Indeed, we are often quite distressed when a repairman returns a machine to us with the words, "I don't know what was wrong with it. I just jiggled it, and now it's working fine." He has confessed that he failed to come to understand the law of the broken machine and we infer that he cannot now know, and neither can we or anyone, the law of the "repaired" machine. If we depend on that machine, we have become servants of a law we cannot know, hence of a capricious law. And that is the source of our distress.

Like all machines, blind justice and punch presses do only what they are made to do.

The machines that populate our world are no longer exclusively, or even mainly, clanking monsters, the noisy motion of whose parts defines them as machines. We have watches whose works are patterns etched on tiny plastic chips, watches without any moving parts whatever. Even their hands are gone. They tell the time, when commanded to, by displaying illuminated numbers on their faces. The rotating mills that once distributed electrical charges to the spark plugs of our automotive engines have been replaced by small black boxes again containing patterns etched on plastic chips, that silently and motionlessly dole out the required pulses. We call these, and a thousand other devices like them, machines too.

This stretching of the connotative range of the word "machine" has two quite separable significances: First, it testifies that the folk wisdom recognizes the essential characteristic of the machine to be its relentless regularity, its blind obedience of the law of which it is an embodiment. And that regularity, as the folk wisdom perceives correctly too, has little to do with material motion. This is the insight which permits people to talk of, say, a bureaucracy or a system of justice as a machine. Second, it reveals an implicit, though very vague, understanding in the folk wisdom of the idea that one aspect of mechanism has to do with information transfer and not with the transmission of material power. The arrival of all sorts of electronic machines, especially of the electronic computer, has changed our image of the machine from that of a transducer and transmitter of *power* to that of a transformer of *information*.

Many other machines have internal components whose functions are primarily to transmit information, even though the over-all function of these machines is to provide mechanical power. Consider, for example, an ordinary four-cycle gasoline engine. It is, of course, a power generator. One of its components is a tappet rod, a straight steel rod whose bottom end rides on a camshaft and whose top end can lift the exhaust valve of the cylinder to which it belongs. As the engine turns the main drive shaft, it also turns the camshaft and the cam on which the tappet rod rides. The tappet rod therefore performs an up-down motion which successively and at just the correct times opens and closes the cylinder's exhaust valve. In simple gasoline engines the tappet rod provides both the power to move the valve and the required timing. But in more complicated engines it acts merely as a signaling device to some other agent that actually manipulates the valve. We can imagine it being replaced by a wire attached at one end to a device which senses when gas is to be expelled from a cylinder and, at the other end, to a motor which suitably actuates the valve. Many modern automobile engines are equipped with electronic fuel injection systems that work very much like this.

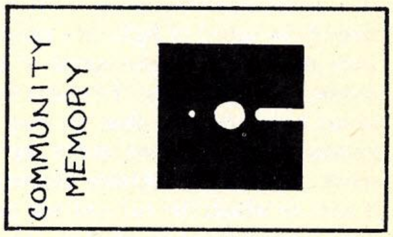
There is, however, a limit to the number of mechanical linkages in an automobile engine that can be replaced by information-transmitting devices. The engine ought, after all, to deliver mechanical power to the wheels of the car. This requirement places the engine's designer under severe constraints. An engineer may very well conceive an internally consistent set of laws, in other words, a design, for an engine that can nevertheless not be realized. His design might require the machining of metals to tolerances that are simply not achievable with the techniques available to him, for example. Or the strengths of the materials required by his engines may not be realizable with the then-available technologies. But much more importantly, his design may be unrealizable in principle because it violates physical law. This is the rock on which, for example, all perpetual-motion machines will always crash. The laws embodied by a machine that interacts with the real world must perforce be a subset of the laws governing the real world.

Monopoly could exist even in worlds in which greed is not a fact.

It is, of course, nonsensical to speak of an embodied machine, one made of material substance, that does not interact with the real world. Were such a thing to exist, we could have no knowledge of it—for, in order for it to give evidence of its existence to us, it would have to affect our senses, hence to interact with the real world. In any case, such a machine would be of no use, for by "use" do we not mean interaction with the world?

But there are circumstances under which it is sensible to speak of aspects of real machines that are separate from the machines' physical embodiments. We sometimes need, for example, to discuss what a machine, or a part of a machine, is to do, quite apart from any consideration of how, or of what materials, one might build a device to actually perform the desired action. For example, some part of a gasoline engine must sense when a cylinder's exhaust valve is to be opened and when closed. That function may be realized by a rigid tappet rod or, as I have said, by a wire suitably connected to a sensor and a motor. The rule that such a device is to follow, the law of which it is to be an embodiment, is an abstract idea. It is independent of matter, of material embodiment, in short, of everything except thought and reason. From such a rule, or "functional specification," as engineers like to say, any number of designs may be evolved, e.g., one may be of a mechanical and another of an electrical "tappet rod." The design of a machine is again an abstraction. A good design, say, of a sewing machine, could be given to several manufacturers, each of whom would produce sewing machines essentially indistinguishable from one another. In a sense then such a good design is an abstract sewing machine. It is the sewing machine which could be manufactured—minus, so to say, the material components, the hardware, of the actual sewing machine. The design is also independent of the medium in which it may be recorded. The blueprint of a machine is not its design. If it were, then the design would change whenever the blueprint is enlarged or redrawn in another color. No, a design is an abstract idea, just as is a functional specification. And ideas, say, the idea of a perpetual-motion machine, are not bound by the laws of physics.

Science-fiction writers are forever coming up with, in effect, functional



FREE
ACCESS

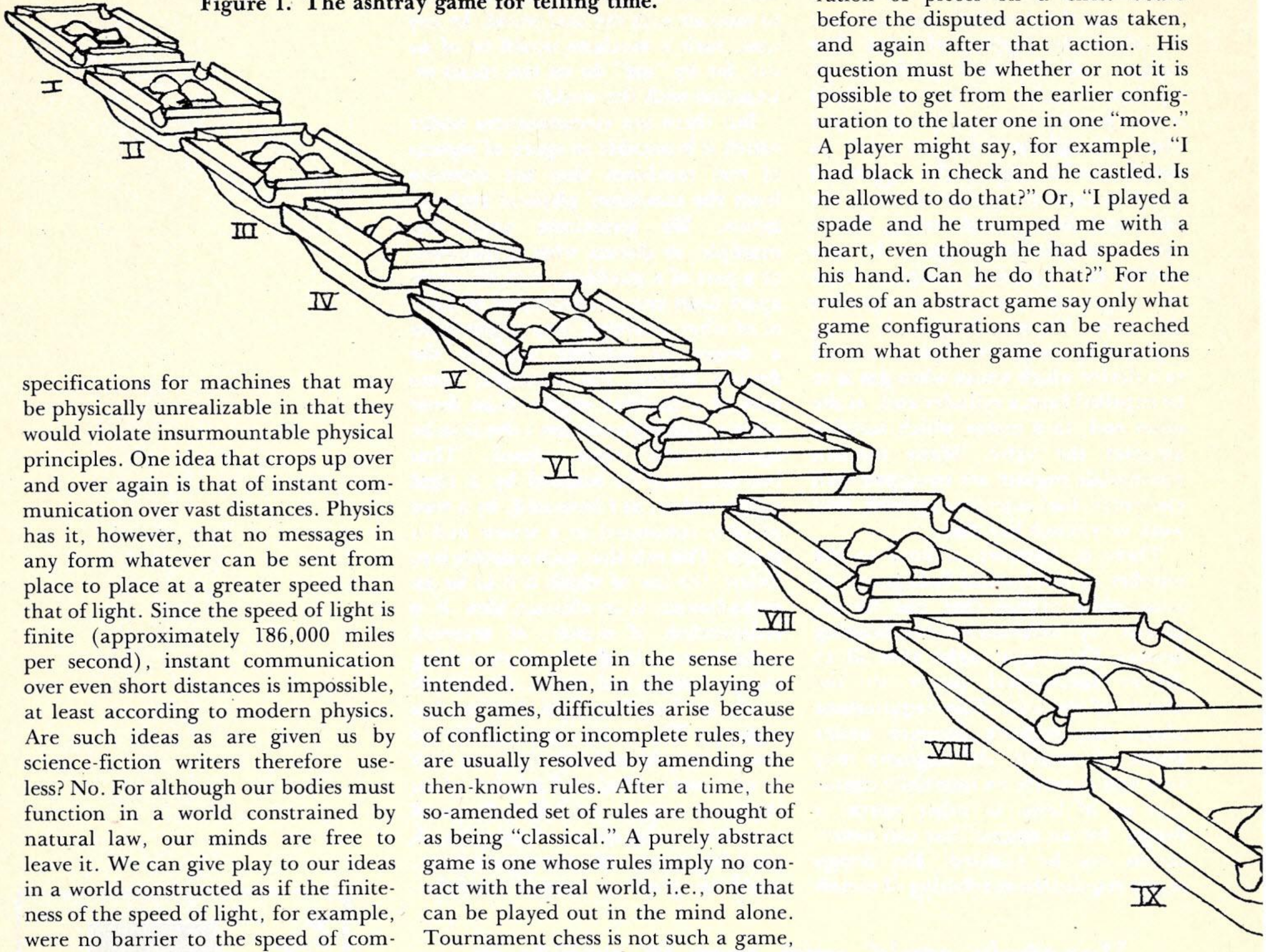


Figure 1. The ashtray game for telling time.

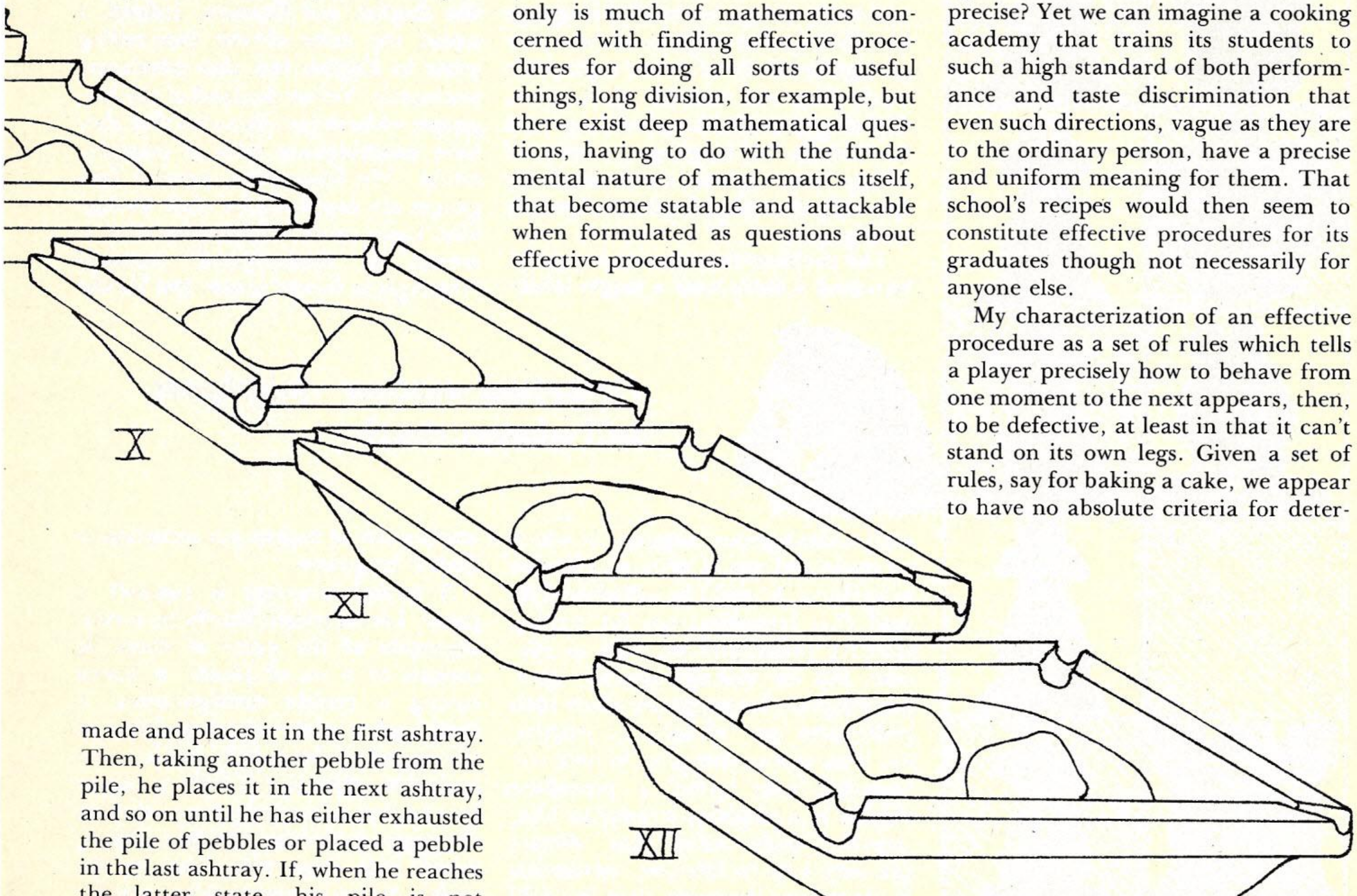
specifications for machines that may be physically unrealizable in that they would violate insurmountable physical principles. One idea that crops up over and over again is that of instant communication over vast distances. Physics has it, however, that no messages in any form whatever can be sent from place to place at a greater speed than that of light. Since the speed of light is finite (approximately 186,000 miles per second), instant communication over even short distances is impossible, at least according to modern physics. Are such ideas as are given us by science-fiction writers therefore useless? No. For although our bodies must function in a world constrained by natural law, our minds are free to leave it. We can give play to our ideas in a world constructed as if the finiteness of the speed of light, for example, were no barrier to the speed of communication generally. To assert that is to say no more than that we may play games whose rules we make up ourselves. We may determine the extent, if any, to which the rules of our games are to correspond to any laws we may think govern the real world. The game "Monopoly" could exist even in worlds, if such there be, in which greed is not a fact.

A crucial property that the set of rules of any game must have is that they be complete and consistent. They must be complete in the sense that, given any proposal for action within the game, they are sufficient for deciding whether that action is legal or not. They must be consistent in the sense that no subset of the rules will determine that a particular action is legal while at the same time another subset determines that that same action is not legal. There are, of course, many games whose rules have never been proved to be either consistent or complete in the sense here intended. When, in the playing of such games, difficulties arise because of conflicting or incomplete rules, they are usually resolved by amending the then-known rules. After a time, the so-amended set of rules are thought of as being "classical." A purely abstract game is one whose rules imply no contact with the real world, i.e., one that can be played out in the mind alone. Tournament chess is not such a game, because its rules limit the amount of time a player can devote to considering his moves. This mention of time puts chess in contact with the real world and thus spoils the purity of its abstractness. Apart from that condition, however, chess is a purely abstract game. Another way to state the condition that the rules of a game must be complete and consistent in the sense here intended, is to say that no two referees faced with the same game situation would fail to agree in their judgment. Indeed, "judgment" is not the proper word, for their decision would be reached by the application of logic only. It would, in effect, be nothing more than a determined calculation, a logical process which could have only one outcome.

There is only one kind of question that could reasonably be given for adjudication to a referee of a purely abstract game. A player could describe the game situation, say, the configuration of pieces on a chess board before the disputed action was taken, and again after that action. His question must be whether or not it is possible to get from the earlier configuration to the later one in one "move." A player might say, for example, "I had black in check and he castled. Is he allowed to do that?" Or, "I played a spade and he trumped me with a heart, even though he had spades in his hand. Can he do that?" For the rules of an abstract game say only what game configurations can be reached from what other game configurations in a single play or move, and, in some cases, what constitutes a winning configuration. We can say this more technically if we speak of a game configuration as a *state of the game* or, even more simply, as a *state,* and of reaching a state from another state as a *state transition.* Using this terminology, we may characterize the rules of an abstract game as *state-transition rules.*

All games that are interesting to play have permissive state-transition rules. The rules permit the player to make one of a sometimes large number of moves when it is his turn to play, except, of course, in relatively rare forced-move situations. Were that not so, the game as such would be pointless; its whole course, hence its outcome, would be determined even before play began. Still, it may be that the outcome, although determined, is not known to the player, and that he desires to know it.

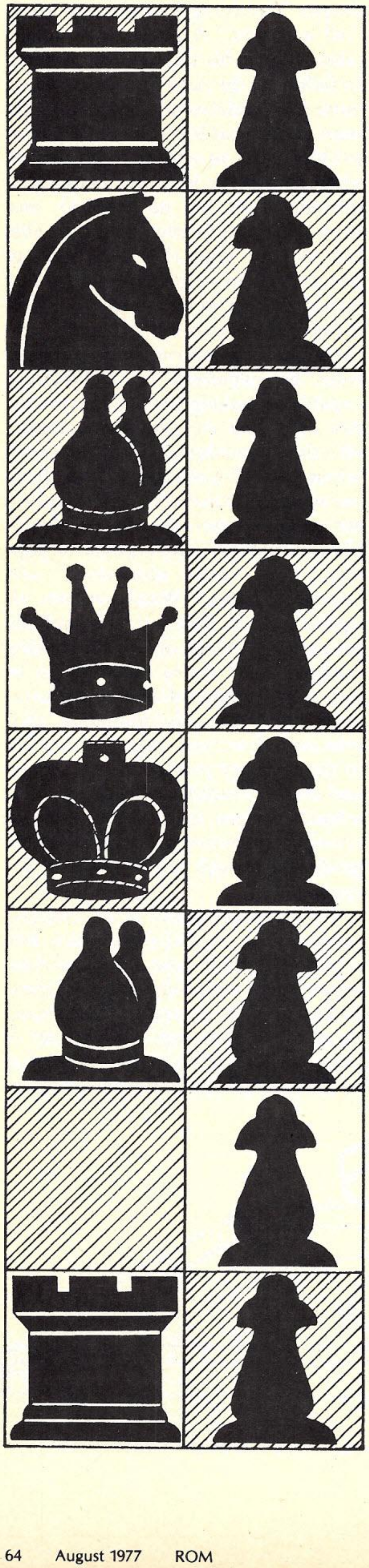
One may wish to know what time it will be, say twenty-two hours after 9 o'clock in the morning. To find that out, one will have to play a simple version of the mathematical game called "modular arithmetic." One way to state the fundamental rule of that game is to say that "x mod z" means the remainder when x is divided by z. In the specific example at hand, our player would want to know what the clock would read twenty-two hours after 9 o'clock, i.e., at 31 o'clock. His problem would be to compute 31 mod 12. (The answer is 7 o'clock.) But let us really make a game of this. The board consists of a set of twelve initially empty ashtrays arranged sequentially (see Figure 1). A large number of pebbles is supplied. The player begins by placing as many pebbles as "what time it is now"—nine in our example—in a pile. He then adds to that pile the number of pebbles corresponding to "the number of hours from now" he has in mind—twenty-two, in our example. He then selects one pebble from the pile he has just made and places it in the first ashtray. Then, taking another pebble from the pile, he places it in the next ashtray, and so on until he has either exhausted the pile of pebbles or placed a pebble in the last ashtray. If, when he reaches the latter state, his pile is not exhausted, he repeats the procedure just described. He will eventually have exhausted the pile he made initially. At this point, the last rule is invoked, namely: if any of the ashtrays are empty, his answer is the number of ashtrays that are not empty; if all ashtrays are empty, his answer is 12 o'clock; but if all ashtrays have at least one pebble in them, he takes one pebble from each ashtray, and then proceeds to apply the last rule again.



Of course, all this is merely a long-winded game for adding two numbers and then dividing their sum by twelve by means of successive subtraction. The rules of the game are not permissive; they don't allow the player to choose the transition from one state of play to the next from a number of alternatives. To the contrary, they command precisely what he must do to make that transition. Such a set of rules—that is, a set of rules which tells a player precisely how to behave from one moment to the next—is called an *effective procedure*. The notion of an effective procedure, or "algorithm," as it is also called, is one of the most important in modern mathematics. Not only is much of mathematics concerned with finding effective procedures for doing all sorts of useful things, long division, for example, but there exist deep mathematical questions, having to do with the fundamental nature of mathematics itself, that become statable and attackable when formulated as questions about effective procedures.

The definition of an effective procedure given above is deceptively simple. The deception is in the words "tell a player." A player who undertakes to solve his time-telling problem by following the rules I have just stated must first understand those rules. He must know what it is to make a pile of pebbles, what an ashtray is, how to tell when an ashtray is empty (suppose it contains ashes, but no pebbles), and so on. He must, in other words, be able not only to read the rules, but to interpret them in precisely the way I intended them to be interpreted. And if the rules are to tell a player "precisely" how to behave, then the rules must be expressed in a language capable of making precise statements. Are cookbook recipes, for example, effective procedures? They certainly attempt to tell a cook what to do from one moment to the next. But then they are generally, even usually, laced with phrases such as "add a pinch of paprika," "stir until consistent," and "season to taste." Would we not all agree that such directions are far from precise? Yet we can imagine a cooking academy that trains its students to such a high standard of both performance and taste discrimination that even such directions, vague as they are to the ordinary person, have a precise and uniform meaning for them. That school's recipes would then seem to constitute effective procedures for its graduates though not necessarily for anyone else.

My characterization of an effective procedure as a set of rules which tells a player precisely how to behave from one moment to the next appears, then, to be defective, at least in that it can't stand on its own legs. Given a set of rules, say for baking a cake, we appear to have no absolute criteria for deter-



mining whether it is or is not an effective procedure. There would be no such difficulty, at least not for cooking recipes, if two conditions were fulfilled: first, that there exists a language in which precise and unambiguous cooking rules could be stated; and, second, that all people are identical in every respect having anything to do with cooking. These conditions are not independent of one another, for one way in which everyone would have to be like everyone else is that they would all have to interpret the cooking language identically. But even the most rigorous cooking academies do not demand that their students become exactly like their master in all relevant respects. They hope only that their graduates have learned to *imitate* the master chef in his interpretation of recipes.

This display of modesty on the part of the senior faculties of cooking schools serves us as an example from which we may learn how to proceed along our own way. In order to give the notion "effective procedure" autonomous status, we need a language in which we can express, without any ambiguity whatever, what a player is to do from one moment to the next. But allegedly effective procedures may be written in languages, many of which are, unless constrained by specially constructed rules, inherently ambiguous.

The problem that thus arises would be solved if there were a single inherently unambiguous language in which we could and would write all effective procedures. It would be sufficient if we used that language, not for writing effective procedures we wish to execute, but for writing rules for interpreting other languages in which such procedures may actually be written. For if an agent competent in only one language were given a procedure written in a language strange to him, together with rules that dictate precisely how to interpret statements in the strange language, then he could imitate what the behavior of a speaker of the strange language would have been had that speaker followed the given procedure. We need therefore some absolutely unambiguous language in which we can write effective procedures and in terms of which we can state rules for interpreting statements in other languages. Such sets of rules would again have to be effective procedures, namely, procedures for the interpretation of sentences of the language to which they apply. But in what language are these rules to be written? We appear to have entered an infinite regress. Had we such a language, and we shall see that we can construct one, we could say of every procedure written in an unambiguous language precisely what it tells us to do: do what the imitating agent does. Hence every such procedure would have a unique interpretation that is independent of the language in which the procedure was originally written.

Perpetual-motion machines will always crash.

I have, by virtue of my silence on the point, let stand the impression that whenever I refer to languages I mean not only formal languages like that of arithmetic, but also natural languages like English and German. Indeed, I stated the rules of the time-telling game in English but also mentioned arithmetic. Yet we demand of the languages we have just discussed that they have unambiguous rules of interpretation. We know that natural languages are notorious for their ambiguity. Later on we will consider what it means to "understand" natural languages in formal terms. But for the moment let us restrict our attention to formal languages.

A formal language is (again!) a game. Let us return briefly to a consideration of the game of chess. It consists of a set of pieces, a board having a certain configuration, a specification of the initial positions of the pieces on the board, and a set of transition rules which tells a player how he may advance from one state of the game to the next. We have already noted that these rules are, except under certain circumstances, permissive; they tell the player the moves he *may* make, but don't dictate what he *must* do. The fact that the initial state of the chess game is specified is a peculiarity of chess, not a reflection of a property of games generally. In poker, players are dealt five cards each, but there is no specification for what these cards must be. Of course, every game must be initialized somehow. We may as well speak of the initialized game as being its starting state, and then add, to the already existing state-transition rules of the game, formation (as opposed to transformation) rules, sometimes permissive as in poker and sometimes mandatory as in chess; formation rules tell how to transform that beginning state into what we ordinarily think of as the initial state of the game, e.g., the cards dealt out or the pieces set up on the board.

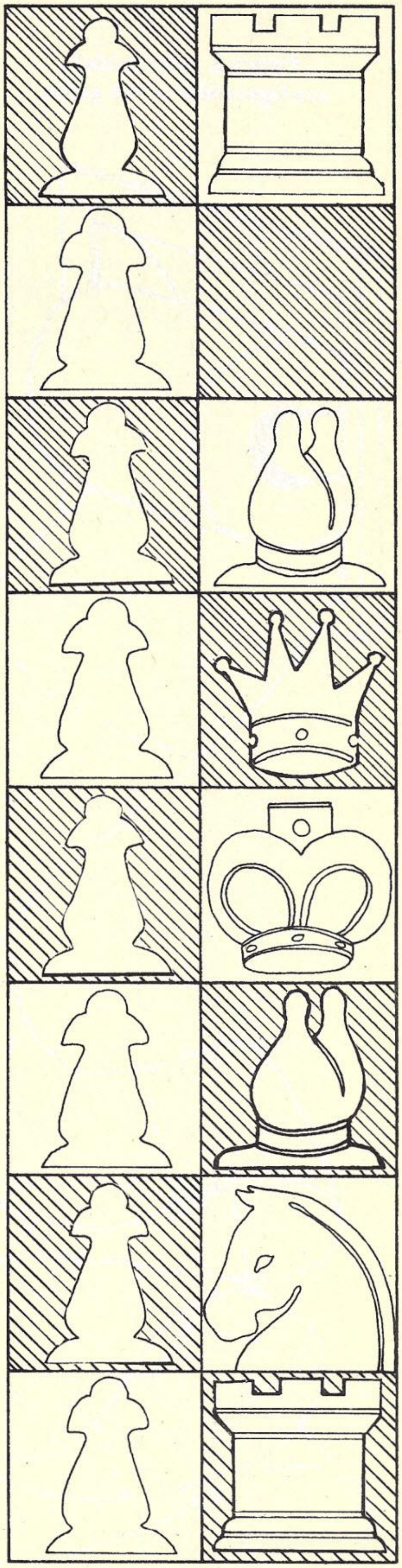
Norwegian sardines in the Second World War were not for eating but for buying and selling.

The "pieces" of a formal language are its *alphabet,* i.e., the set of symbols which may be manipulated in the language. We may, if we wish to preserve the analogy to chess, think of the paper on which the symbols of the language are written as the "board," but that is not important. The transition rules of a formal language play the same role for it as the transition rules of a game play for the game: they tell a player how to move from one state of the game to another. I said earlier that the only significant question that can be put to a game's referee is whether a proposed move is legal or not. Exactly the same is true for formal languages. However, with formal languages, although strictly speaking only that one question is possible, it can take two different forms: first, "Is the proposed transition legal?" and second, "Is the configuration of symbols under consideration an admissible expression in the language?" "May I castle while my king is in check?" is a chess question of the first form. "Is the board configuration here exhibited one that could possibly be reached by legal play?" would be a chess question of the second form. For some board configurations that question is easy to answer. Were we asked it about a board, for example, on which there were eight white pawns and on which two white bishops occupied squares of the same color, we would answer "No." Similarly if we found two kings of the same color on the board, and so on. But such questions are simply not asked by chess players. The reason for this is that a chess game always starts from the standard board configuration or is resumed from a position achieved by a temporarily suspended game.

Many formal languages differ from chess in this respect. High-school algebra—whose rules I will not detail here—has, for example, transformation rules for factoring algebraic expressions; e.g., $ac+bc$ is transformed into $(a+b)c$ by one such rule. But in order for such rules to be applicable at all, the expressions to which they are to be applied must first of all be legal (grammatical) expressions (or sentences) in the language. The expression $ac+bc+$ is not a correct sentence in algebra and none of algebra's transformation rules apply to it. If one is to play algebra, then, one must first set up the board in a legal manner. One must know that an expression beginning with a left parenthesis must somewhere be "closed" by a matching right parenthesis, that operator symbols like "+" must be placed between two expressions, and so on.

A formal language is a game. That is not a mere metaphor but a statement asserting a formal correspondence. But if that statement is true, we should, when talking about a language, be able easily to move back and forth between a game-like vocabulary and a corresponding language-like vocabulary. Precisely that can be done.

I will describe a game in terms of a board, pieces, moves, and so on, and finally develop the corresponding linguistic notions. My purpose is to develop a very precise language on a very small alphabet, moreover, a language whose transformation rules can also be written using only that small alphabet. I will sketch the design of a simple machine embodying those



transformation rules. That machine will, of course, be able to play the game I described initially. I will then write the rules of the game I have designed in a language whose alphabet corresponds to the game's pieces. It is that language that is the game. Finally, I will, indicate how a second machine can be designed to function as an interpreter of this new language. We shall see where we go from there.

THE GAME

Equipment:

One roll of toilet paper.

Many white stones, five black stones, and an ordinary six-sided playing die.

Initialization:

1. Roll out the toilet paper on the floor.
2. Put down stones as follows:
   - i. on an arbitrary square, one black stone;
   - ii. on successive squares to the right of the square holding the black stone; as many white stones as you please, one to each square;
   - iii. on successive squares continuing to the right, one black stone, skip one square, one black stone;
   - iv. on successive squares continuing to the right, an arbitrary number of white stones;
   - v. on the square to the right of the last white stone, one black stone;
   - vi. finally, one black stone, the "marker," above the square holding the rightmost white stone.
3. Turn the die so that its one-dot side is facing upward, i.e., so that it is showing "1."

(An example of such a setup is shown in Figure 2.)

The Transformation Rules:

The marker stone is moved either one square to the left or one to the right on each move. However, before each move, the stone under the marker stone is replaced or removed according to the applicable rule. The die may be turned to a new side after each move.

The rules govern: what kind of stone, if any, is to replace the stone under the marker; what side of the die is to be turned up; and in which direction the marker is to be moved. There are eighteen rules. They all have the same form. Each of them describes an

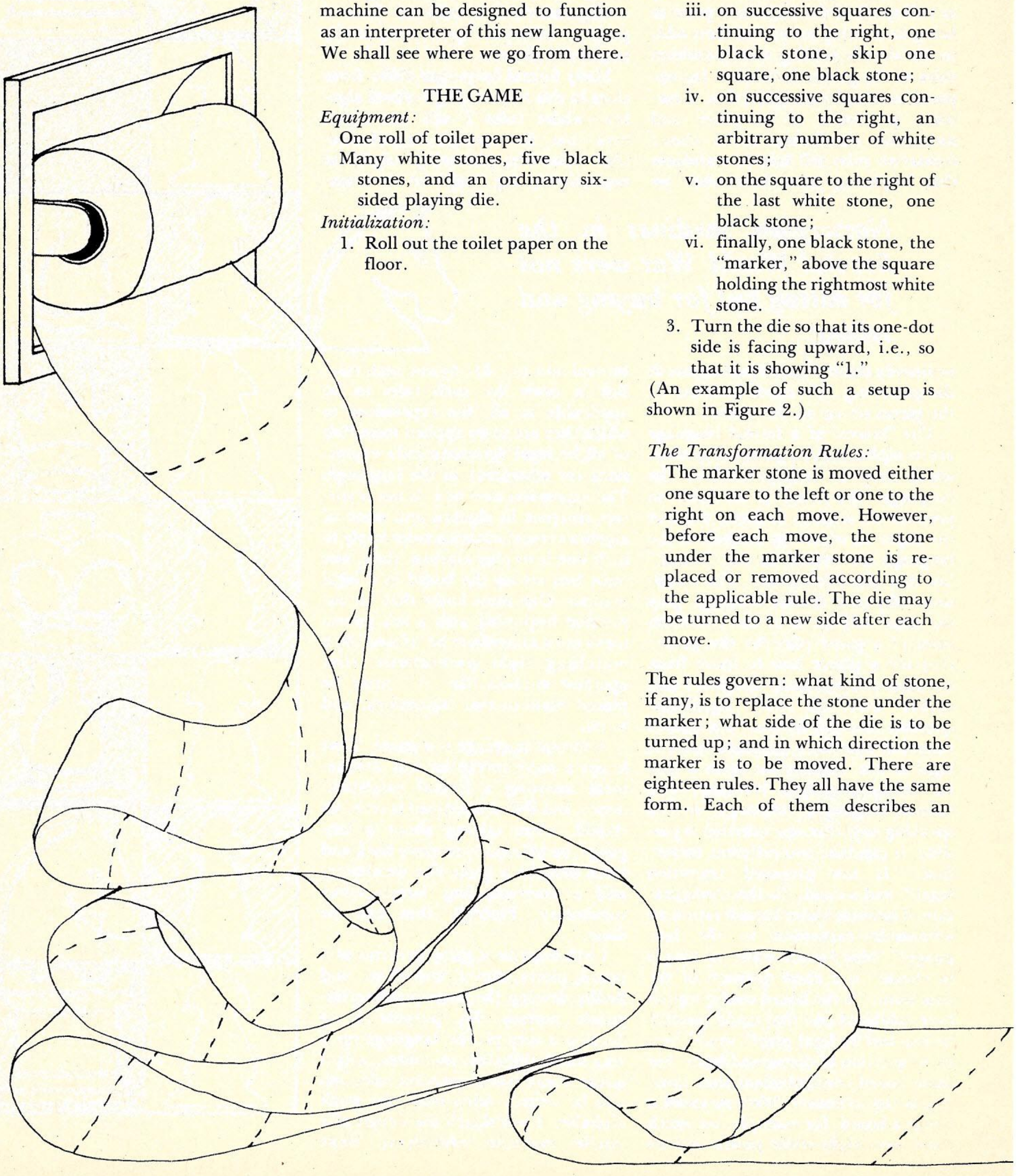


Figure 2. The initial configuration of the game.

orientation of the die, and a specific kind of stone or no stone at all under the marker. The player must find the rule that corresponds to the existing game situation, i.e., the situation defined by what the die reads and by what kind of stone, possibly no stone at all, is under the marker. He is then to do what the rule tells him to do. Each rule says to do three things:

1. Turn the die so that it reads the stated number.
2. Replace the stone under the marker by the kind of stone specified—possibly by no stone at all.
3. Move the marker one square in the indicated direction.

Obeying a rule thus creates a new game situation. The player again applies the rule then appropriate, and so on. When a rule tells him to turn the die so that it will read "0," the game stops.

The rules are given in the form of a table (Table 1). The first two columns describe the conditions under which the corresponding rule is applicable, and the remaining three columns of the corresponding row tell what to do.

Table 1. The Rules of the Game.

IF THE DIE READS	AND THE STONE UNDER THE MARKER IS	THEN TURN THE DIE TO	REPLACE THE STONE BY	MOVE MARKER
1	none	3	white	left
1	black	2	no stone	left
1	white	1	white	left
2	none	2	no stone	left
2	black	3	no stone	left
2	white	5	no stone	right
3	none	3	no stone	left
3	black	4	no stone	right
3	white	5	no stone	right
4	none	4	no stone	right
4	black	1	black	right
4	white	6	white	left
5	none	5	no stone	right
5	black	1	black	right
5	white	1	white	left
6	none	0	no stone	right
6	black	0	black	right
6	white	3	white	left

Of course, just as Norwegian sardines in the Second World War were not for eating but for buying and selling, so this game is not for playing but for talking about. In order to be able to talk about it more easily, let us change notation: instead of "black," "white," and "no" stones, we will use "X," "1," and "0," respectively. An initial board configuration is then

....000X11X0X11$\underline{1}$X00....,

where "...." means "and so on." (Remember, the whole roll of toilet paper contains "0," i.e., no stones, initially.) The marker is really only an aid to memory; it can be replaced by the player's index finger. I have here underlined the marked place. If we now interpret rows of 1's as numbers—"111" means "3"—we can see that the X's serve as punctuation, rather like quotation marks, enclosing two numbers. Given the above initial configuration, the play would end with the configuration

....00$\underline{0}$11111X00....

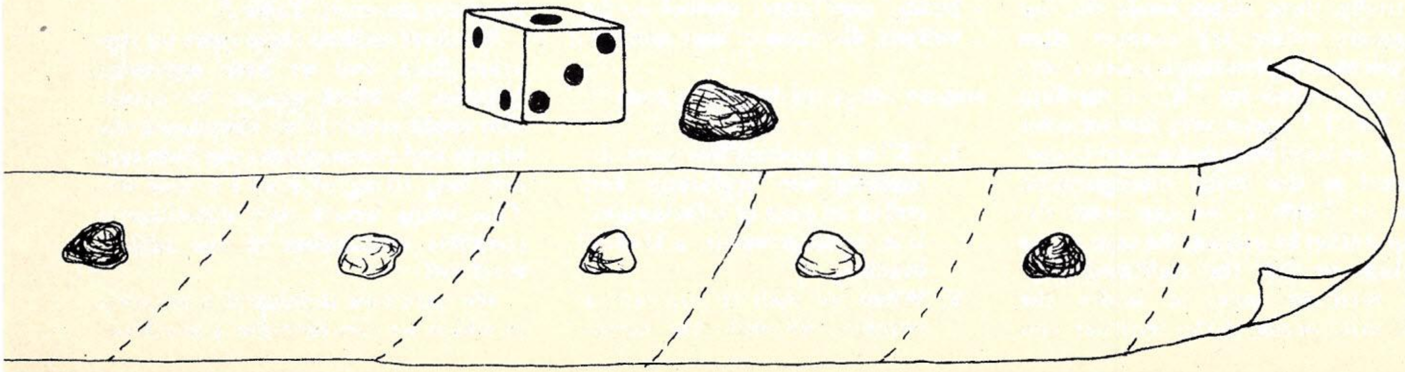
This configuration may be interpreted as the sum of the two numbers initially presented. The game thus constitutes (although I haven't here proved it) an adding machine. The whole game is shown in Table 2, in which the rightmost column gives the number displayed by the die.

As an example of an application of a rule, consider the transition from line 6 to line 7 in the game shown. Line 6 is

X11$\underline{\text{X}}$00111X

with the die reading 2. The applicable rule must therefore say "If the die reads 2 and a stone of kind X, i.e., a black stone, is under the marker, then do such and such." The fifth—and only the fifth—of the given rules matches the conditions pertaining to line 6. It says to do as follows:

1. Turn the die so that it reads 3.
2. Replace the black stone under the marker by no stone; i.e., remove the black stone.
3. Move the marker one square to the left.



After that rule has been followed, the board configuration of the game is then the one displayed in line 7,

X1$\underline{1}$000111X,

and the die is left reading 3.

Now look at the play of this game as a whole, and notice particularly how the marker shuffles back and forth. To aid the intuition, think of the game being played on a field. The stones are very heavy and the boy moving them must rest each time he moves from one position to the next. We can see the major outlines of his strategy: He searches for the rightmost "1" of the left number (and finds it in step 7) and then for a place to put it. To find that, he must find the leftmost "1" of the right number (which he finds in step 11) and replace its left neighbor by "1." He continues in this fashion until he runs into the leftmost "X," a boundary marker, and concludes he is done. Of course, he may not see this strategy. He carries the rules with him and consults them between each move.

It is easy to see how we could build a machine to do this work. We replace the roll of toilet paper with an ordinary reel of magnetic tape, and the player with an ordinary tape recorder. Of course, we have to change the rules a little: now the tape moves, not a marker along the tape. Therefore whenever we specified a marker motion to the left, we must now specify a tape motion to the right, and vice versa. The three symbols we need can be represented by three easily distinguishable tones. We install six relays in the tape recorder to hold the information formerly conveyed by the die. (Actually, three relays would do, but imaginary relays are cheaper than complicated explanations.) Let a very high tone stand for "X," a medium tone for "1," and a very low tone for "0." If we have recorded a tape to correspond to the initial configuration given in Table 2, we can start the computation by placing the tape in the machine so that the rightmost "1," i.e., medium tone, is under the read/write heads of the recorder and by seeing to it that relay 1 is closed—because the die showed 1 initially—and that all other rule relays are open. The circuitry of the machine is so arranged that:

> If relay 1 is closed and if a high tone (X) is read, then the high tone is removed by overrecording a low tone ("0"), the tape is moved one square to the right (this effectively moves the recorder's heads, i.e., the marker, to the left), relay 1 is turned off (opened), and relay 2 is turned on (closed).

Given these rules, it is an adding machine—given some others, it would be quite a different machine.

This is a faithful translation of the second line of our original table of rules. Similarly, every other one of our rules is translated into tape-recorder terms. We then have an incredibly awkward adding machine.

The most important thing to notice about this machine is that it is completely defined by the rules of the game as translated into tape-recorder terms. Given those rules, it is an adding machine; given some others, it would be some other kind of machine.

Now notice also that we needed only three distinguishable symbols on our tape. We denoted them by "X," "1," and "0." It turns out that we can write our entire set of rules using only these three symbols as well!

Let us speak of the "state" of the machine as being the number showing on its die, i.e., the number of the rule relay that happens to be on (closed). We now write the five parts of our rules in the following sequence,

> Current state, symbol under heads, next state, symbol to be written, direction of tape motion,

and we adopt the following code:

1. "X" is a punctuation mark indicating the beginning and end of an item of information. It is, in other words, a kind of bracket.
2. When we wish to indicate a number, we write the corresponding number of 1's.
3. In the context "Direction of tape motion," "0" stands for "left" and "1" for "right."

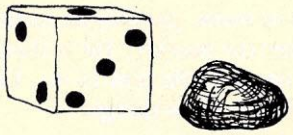


Table 2. The Game.

STEP	BOARD	NEXT RULE
1	X11X0X11$\underline{1}$X	1
2	X11X0X1$\underline{1}$1X	1
3	X11X0X$\underline{1}$11X	1
4	X11X0$\underline{X}$111X	1
5	X11X$\underline{0}$0111X	2
6	X11$\underline{X}$00111X	2
7	X1$\underline{1}$000111X	3
8	X10$\underline{0}$00111X	4
9	X100$\underline{0}$0111X	5
10	X1000$\underline{0}$111X	5
11	X10000$\underline{1}$11X	5
12	X1000$\underline{0}$111X	1
13	X100$\underline{0}$1111X	3
14	X10$\underline{0}$01111X	3
15	X1$\underline{0}$001111X	3
16	X$\underline{1}$0001111X	3
17	X0$\underline{0}$001111X	5
18	X00$\underline{0}$01111X	5
19	X000$\underline{0}$1111X	5
20	X0000$\underline{1}$111X	5
21	X000$\underline{0}$1111X	1
22	X00$\underline{0}$11111X	3
23	X0$\underline{0}$011111X	3
24	X$\underline{0}$0011111X	3
25	$\underline{X}$00011111X	3
26	0$\underline{0}$0011111X	4
27	00$\underline{0}$011111X	4
28	000$\underline{0}$11111X	4
29	0000$\underline{1}$1111X	4
30	000$\underline{0}$11111X	6

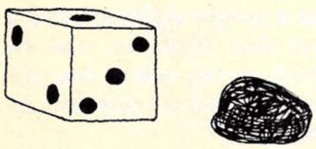
Using this code we may now transliterate the rules given in Table 1 into the form shown in Table 3.

We have written these rules on separate lines, and we have separated columns by blank spaces. No confusion could result if we eliminated the blanks and concatenated the lines into one long string of X's, 1's, and 0's. That string would then constitute a complete description of our adding machine!

We have now developed a notation in which we can describe a machine.

Its alphabet consists of the three symbols "X," "0," and "1." Of course, strings written in this notation would remain meaningless unless we could also say how they are to be interpreted. To do this, it would be sufficient to describe a machine—or, better yet, to build one—that would take as its input the two streams of information consisting of, first, the properly encoded description of our adding machine, and, second, an initial configuration of X's, 0's, and 1's on which the so-described adding machine is to operate. These two so-called *strings* of symbols could, of course, be put on a single tape. Once we have such a machine, we are entitled to call our system of notation a language, for we will then have an embodiment of its transformation rules.

In one of the greatest triumphs of the human intellect, the English mathematician Alan M. Turing proved in 1936 that such a machine could be built and even showed how to build it. He actually proved much more—but of that, more later. I cannot here describe a machine built according to Turing's principles in any great detail, but I must say a few words about it.



Imagine again a tape recorder such as the one we used for our adding machine. Again it has a set of relays capable of representing its states. This time, because the machine is much more complicated than our adding machine, there are many more states to be represented and hence more relays—but that is a detail. The tape we give our new machine has the following layout; reading from right to left:

- a section containing the description of our adding machine in the notation we have developed;
- a section containing the data on which the adding machine is to work, for example, "X11X0X111X";
- a section that can store the current state of the "adding machine";
- an arbitrarily long section of "blank" tape, i.e., a section containing 0's.

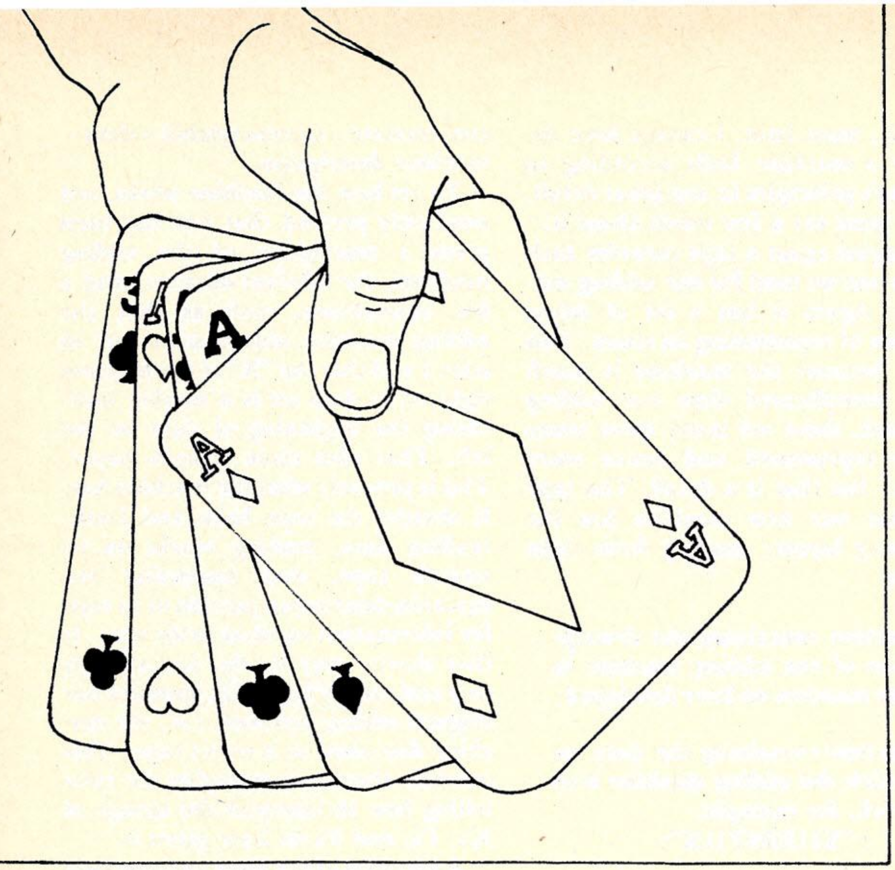
The information on the tape then has the following structure: blank tape—current state—current symbol—data—machine description.

To see how the machine works, one need only pretend that one has been given a description of the adding machine, the relevant data set, and a few conventions, such as that the adding machine always starts off in state 1 and that the "X" at the extreme right of its data set is a marker indicating the beginning of data to the left. That plus some scratch paper. This is precisely what the machine has. It shuttles the tape back and forth, reading data, making marks on its scratch tape, and inspecting the machine-description portion of its tape for information on what to do next. It thus slowly, very slowly, but step by step and utterly faithfully, imitates our original adding machine, i.e., the machine described to it on its tape. This machine then truly embodies the rules telling how to interpret the strings of X's, 1's, and 0's we have given it.

A machine of the kind we have been discussing, i.e., a machine that shuttles a tape back and forth, reading and changing marks on a square of tape at a time and going from one state into another, and so forth, is today called a Turing machine. Such a machine is completely described when, for every state it can attain and for every symbol that can be under its reading head, the description states

Table 3. The Rules of the Game.

RULE NUMBER	SYMBOL UNDER HEAD	NEXT RULE NUMBER	SYMBOL TO WRITE	DIRECTION OF TAPE MOTION
X1X	X0X	X111X	X1X	X1X
X1X	XXX	X11X	X0X	X1X
X1X	X1X	X1X	X1X	X1X
X11X	X0X	X11X	X0X	X1X
X11X	XXX	X111X	X0X	X1X
X11X	X1X	X11111X	X0X	X0X
X111X	X0X	X111X	X0X	X1X
X111X	XXX	X1111X	X0X	X0X
X111X	X1X	X11111X	X0X	X0X
X1111X	X0X	X1111X	X0X	X0X
X1111X	XXX	X1X	XXX	X0X
X1111X	X1X	X111111X	X1X	X1X
X11111X	X0X	X11111X	X0X	X0X
X11111X	XXX	X1X	XXX	X0X
X11111X	X1X	X1X	X1X	X1X
X111111X	X0X	XX	X0X	X0X
X111111X	XXX	XX	XXX	X0X
X111111X	X1X	X111X	X1X	X1X



what symbol it is to write, what state it is to go into, and in which direction it is to move its tape. We can stipulate, as a matter of convention, that every such machine always start in its first state, call it state 1, and with the tape so positioned that its rightmost symbol is under the read/write head. We happened here to use a language whose alphabet consists of the three symbols we used to describe our adding machine. Had we been more generous, i.e., had we permitted ourselves the use of a larger alphabet, our machine could have been simpler, in the sense of having fewer states. On the other hand, the rules would have been more complicated. There are, then, many possible realizations of our adding machine—at least one for every substantially different alphabet we could have chosen. The minimum size of an alphabet we could use is two—an adding machine such as ours would have to have many states to operate with such a restricted alphabet. The minimum number of states such an adding machine would have to have is two—but this machine would have to operate on a large alphabet.

Every modern computer is a universal Turing machine.

There is an important difference between the two machines we have been discussing: our adding machine is a special-purpose machine. It can add any two numbers, but it can do nothing else. The second machine requires as input an encoded description of some machine and a data set on which the described machine is to operate. In effect, the machine description it is given is a *program* which transforms the second machine into the machine it is to imitate. The question naturally arises "What kinds of machine can be imitated in this way?"

I have illustrated what I mean by a formal language, namely: an alphabet; a set of formation rules that determine the format of strings of symbols constructed on that alphabet, that constitute legal expressions in the language; and a set of transformation rules for such expressions. I have also said that it is possible to construct machines that embody such transformation rules and that can therefore execute procedures represented in the corresponding languages. Beyond that, I have discussed a machine that accepts descriptions of other machines and is capable of imitating the behaviors of the described machines. We have thus gained an idea of what is meant by imitation in the present context. Note carefully that I have never alluded to translation of one language to another. Our imitating machine does not first translate the transformation rules we gave it, i.e., the encoded description of our adding machine, into its own or any other language. It consults—we use the word "interpret"—that set of transformation rules each time it must decide what the machine it is imitating would do. It thus makes many moves for every move the imitated machine would have made.

My aim, for the moment, is to put a firm foundation under our concept of effective procedure. We wish for a single language in terms of which effective procedures can be expressed, at least in the sense that we can describe all our procedural languages in that language and thus give our procedures unique interpretations. We can now see that the transformation rules of languages can be embodied in machines. My task is therefore reduced to showing that a unique alphabet, and a language on that alphabet, can be found in which we can indeed describe all languages in which we may want to write procedures.

We can design a language whose alphabet consists of only two symbols, say, "0" and "1," in terms of which we can describe any Turing machine. Now we have seen that a language consists of more than an alphabet, just as a game consists of more than the pieces with which it is played. Its transformation rules must also be given. In the present context I intend the transformation rules of the language I have in mind to be embodied in a Turing machine similar to the imitating machine we have already discussed. I am saying, then, that there exists a Turing machine that operates on a tape containing only 1's and 0's and that is capable of imitating any other Turing machine whatever. This so-called *universal* Turing machine is, as are all Turing machines, describable in terms of a set of quintuples of the form with which we are already familiar, i.e.,

(current state, symbol read,
next state, symbol written,
direction of tape motion),

and these quintuples, in turn, may be written in the language that that Turing machine is designed to accept. The universal Turing machine is consequently capable of accepting a description of itself and of imitating itself.

In fact, one can design many languages with the same two-symbol alphabet, i.e., many universal Turing machines embodying rules of transformation on strings of these two symbols. And one can, of course, enlarge the alphabet and design many universal Turing machines corresponding to each such enlargement. But it is the principle that interests us for the moment, namely, that

> there exists a Turing machine *U* (actually a whole class of machines) whose alphabet consists of the two symbols "0" and "1" such that, given any procedure written in any precise and unambiguous language and a Turing machine *L* embodying the transformation rules of that language, the Turing machine *U* can imitate the Turing machine *L* in *L's* execution of that procedure.

This is a restatement of one of the truly remarkable results that Turing announced in his brilliant 1936 paper.

There are many existence proofs in mathematics. But there is a vast difference between being able to prove that something exists and being able to construct it. Turing proved that a universal Turing machine exists by showing how to construct one. We have to remember that Turing did this monumentally significant work in 1936—about a decade before the first modern computers were actually built. Modern computers hardly resemble the machine Turing described. Many have, for example, the ability to manipulate many magnetic tapes simultaneously and, even more importantly, most are equipped with very large information stores. The storage mechanism of a modern computer is functionally like a set of relays, each of which can be either on (closed) or off (open). A set of ten such relays can take on 1,024 different states. It is not uncommon for a modern computer of moderate size to have more than a million such elementary storage components, and thus to be able to take on $2^{1,000,000}$ states. That is an unimaginably huge number. (The Earth, for example, weighs much less than $2^{1,000}$ pounds.) Still, in principle, every modern computer is a Turing machine. Moreover, every modern computer, except for a very few special-purpose machines, is a *universal* Turing machine. And that, in practice, means that every modern computer can, at least in principle, imitate every other modern computer.

Granting that any computer can do what any other computer can do, there remains the question of what computers can do at all.

There is still one more hole to be plugged. For even granting that, in effect, any computer can do what any other computer can do, there remains the question of what computers can do at all, i.e., for what procedures one can realize Turing machines, hence Turing machines imitable by universal Turing machines, and hence imitable by modern computers. Turing answered that question as well: a Turing machine can be built to realize any process that could naturally be called an effective procedure.

This thesis, often called Church's thesis after the logician Alonzo Church, who formulated it in a framework different from Turing's, cannot be proven, because it involves the word "naturally." In a sense, we are stuck in a logical circle; any process we can describe in terms of a Turing machine is an effective procedure, and vice versa. What lends real intuitive strength to the idea, however, is the fact that several radically different and independently derived formulations of the idea of "effective computability" have all been shown to be equivalent to computability in Turing's formalism and hence to one another. As M. Minsky remarks: "Proof of the equivalence of two or more definitions always has a compelling effect when the definitions arise from different experiences and motivations."

But even though we must rely on our intuition as to what may "naturally" be called an effective procedure, we are now on firm ground in being able to say precisely and unambiguously what an effective procedure tells us to do. At least in principle, we can encode the

alphabet of the language in which the procedure is written, using only the two symbols "0" and "1." We can then transcribe the rules that constitute the procedure in the new notation. Finally, we can given some universal Turing machine that operates on the 0,1 alphabet the transformation rules of the procedure's language in the form of suitably encoded quintuples. The given procedure tells us to do what the so-instructed universal Turing machine does as it imitates the "machine" we have described to it. If we understand how a Turing machine operates at all (and such understanding involves very little knowledge, as we have seen), and if we have a description of the universal Turing machine we appealed to, then we know what the procedure tells us to do in detail.

Such a way of knowing is very weak. We do not say we know a city, let alone that we understand it, solely on the basis of having a detailed map of it. Apart from that, if we understand the language in which a procedure is written well enough to be able to explicate its transformation rules, we probably understand what rules stated in that language tell us to do.

But such objections, valid as they are, miss the point. Turing's thesis tells us that we can realize, as a computer program, any procedure that could "naturally" be called an effective procedure. Therefore, whenever we believe we understand a phenomenon in terms of knowing its behavioral rules, we ought to be able to express our understanding in the form of a computer program. Turing proved that all computers (save a few special-purpose types that do not concern us here) are equivalent to one another, i.e., are all universal. Hence any failure of a technically well-functioning computer to behave precisely as we believe we have programmed it to behave cannot be attributed to any peculiarity of the specific computer we have used. Indeed, the fault must be that we have been careless in our transcription of the behavioral rules we think we understand into the formal language demanded by our computer, or must be in the initial explication, in any form, of what we had in mind when we believed we understood, or must be that our understanding is defective. The last is most often the case. I shall say much more about that later. For now, we need note only that the defect in our understanding can take two forms:

First, although our theory may be on the whole correct, it may contain an error in detail. We wrongly assert, for example, that if this and that is true, then so-and-so follows. Our mental processes, lulled perhaps by the sheer eloquence of the argument we make to ourselves, often permit us to slide over such errors without the slightest disturbance. The computer is, however, very unforgiving in this respect. It follows the logic we have given it. That logic may lead to very different consequences than do mental processes contaminated by wishes to reach certain outcomes. Indeed, one of the most cogent reasons for using computers is to expose holes in our thinking. Computers are merciless critics in this respect.

One of the most cogent reasons for using computers is to expose holes in our thinking.

Second, the defect in our understanding may be that, although it is true that we understand, we are still not able to formalize our understanding. We may, for example, be able to predict with very great confidence what an animal will do under a large variety of circumstances. But our predictive power, great and reliable as it may be, may rest on intuitions that we are simply unable adequately to explicate. Yet we may be driven to force our ideas into a formal mold anyway. A computer program based on a formal system so derived is certain to misbehave. The trouble then is not merely that the theory it represents contains certain errors in detail, but that that theory is grossly wrong in what it asserts about the matters it concerns. It is not always clear which defect one is confronted with when a computer one has programmed misbehaves. There is usually enormous motivation to believe that one's theories are all right on the whole, and that, when they don't work well, there must be some error in detail that can easily be patched up. I shall have more to say about such matters later.

We have used the idea of the universality of computers in the foregoing. We must now ask whether the universality of computers implies that they can "do anything." This is really the question "Can anything we may wish to do be described in terms of an effective procedure?" The answer to that question is "No."

First, there are certain questions that can be asked and for which it can be proved that no answers can be produced by any effective procedure whatever. We may, for example, be interested to know whether some machine we have designed, say, our adding machine, will halt once started with a particular data set. It would be convenient if we had a testing machine which could, for any machine and any data set appropriate to it, tell us whether that machine operating on the given data set would ever halt. No such machine can be built. This and many other such "undecidable" questions therefore impose some limit on what computers can do. Of course, this is a logical limitation, which constrains not only electronic computers but every computing agent, human and mechanical. It has also to be said that the whole set of undecidable questions is not terribly interesting from a practical point of view; all such questions are vastly general. If we had some specific computation about which we wanted to know whether or not it would ever terminate, we usually could design a procedure to discover that. What is impossible is to have a machine—or, what is the same thing, an effective procedure—that will make that discovery for any procedure *in general.*

Second, an effective procedure may be capable of making some calculation in principle, but may take such a long time to complete it that the procedure is worthless in practice. Consider the game of chess, for example. Given the rule that a game is terminated if the same board configuration is achieved three times, chess is certainly a finite game. It is therefore possible, in principle, to write a procedure to generate a list of all games, move for move, that could possibly be played. But that computation would take eons to complete on the fastest computers imaginable. It is therefore an example of an impractical procedure. Indeed, we have discussed procedures up to this point as if the time they take to do their work, i.e., to complete their computational task, were irrelevant. Such an attitude is appropriate as long as we are in the context of abstract games. In practice, of course, time does make a difference. We must note in particular that, when one computer is imitating another, it must go through many time-consuming steps for every single step of the imitated computer. Were that not so, we would strain to build the cheapest possible universal Turing machine and, since it could imitate every more expensive machine, it would soon drive all others from the market.

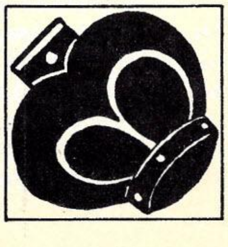
Third, we may write a procedure realizable by a Turing machine, hence an effective procedure, but one whose rules do not include an effective halting rule. The procedure, "beginning with zero, add one, and, if the sum is greater than zero, add one again, and so on," obviously never stops. We could substitute "if the sum is less than zero, stop, otherwise add one again" for "if the sum is greater than zero, add one again" in that procedure and thus provide it with a halting rule. However, a computation following that procedure would never encounter the halting rule, i.e., the corresponding Turing machine would never fall into the state corresponding to "sum less than zero." The procedure is therefore, in a sense, defective. It is not always easy, to say the very least, to tell whether or not a real procedure written for real computers is free of defects of this and similar kinds.

Finally we come to the most troublesome point concerning what computers can and cannot do. I have said over and over again that an effective procedure is a set of rules which tells us in precise and unambiguous language what to do from one moment to the next. I have argued that a language is precise and unambiguous only if its alphabet and its transformation rules can themselves be explicated in precise and unambiguous terms. And I have repeated Church's (and Turing's) thesis that, to every such explication of whatever language, there corresponds a Turing machine that can be imitated by a universal Turing machine. I have asserted further that virtually every modern computer is a universal Turing machine. Leaving to one side everything having to do with formally undecidable questions, interminable procedures, and defective procedures, the unavoidable question confronts us: "Are all the decisionmaking processes that humans employ reducible to effective procedures and hence amenable to machine computation?"

Are all decision-making processes that humans employ amenable to machine computation?

We have seen that the very idea of an effective procedure is inextricably tied up with the idea of language. Isn't it odd that I could have spent so much time discussing language without ever alluding to *meaning?* The reason I have been able to avoid confronting the concept of meaning is that I have been discussing only formal languages or, as I have said, abstract games. Not that meaning plays no role whatever in such language games. It does. But this role is entirely subsumed in the transformation rules of the language. Recall that in algebra we may transform $ac+bc$ into $(a+b)c$. We are entitled to say that the two expressions mean the same thing, or, to put it another way, that the transformation we have employed preserves the "value" of the original expression. In still other terms, were we to substitute numbers for a, b, and c, the two expressions would both produce the same result upon execution of the indicated arithmetic operations. (This last is, by the way, not a property of all algebras. Elementary algebra has been deliberately designed so that its transformation rules are consistent with those of the formal language we call arithmetic.) It is a property of formal languages, indeed, it is their essence, that all their transformation rules are purely syntactic, i.e., describe permissible rearrangements of strings of symbols in the language, including replacements of symbols and introductions of new symbols—e.g., ")" and "("—independent of any interpretation such symbols may have outside the framework of the language itself. One can, for example, do pages of algebraic transformations, following the rules of algebra blindly, without ever having to know that one may substitute numbers for lowercase letters but not for parentheses, in other words, without ever giving any interpretations to the symbols one is dealing with. The same is not true for natural language. Consider the English sentence: "I never met a man who is taller than John." It may be transformed into "I never met a taller man than John." This transformation clearly preserves the meaning of the original sentence. But if we apply the same transformation rule to "I never met a man who is taller than Maria," and get "I never met a taller man than Maria," it no longer works. The rule we have applied is not purely syntactic. It concerns itself not merely with the *form* of uninterpreted strings of symbols, but with their *meanings* as well.

We have seen that, at a certain level of discourse, there is no essential difference between a language and a machine that embodies its transformation rules. We have also noted that, although the laws of which abstract machines are embodiments need not necessarily be consistent with the laws of the physical universe, the laws em-



bodied by machines that interact with the real world must perforce be a subset of the laws governing the material world. If we wish to continue to identify languages with machines even when discussing natural language, then we must recognize that, whatever machines correspond to natural languages, they are more like machines that transform energy and deliver power than like the abstract machines we have been considering; i.e., their laws must take cognizance of the real world. Indeed, the demands placed on them are, if anything, more stringent than those placed on mere engines. For although the laws of engines are merely subsets of the laws of physics, the laws of a natural-language machine must somehow correspond to the inner realities manifest and latent in the person of each speaker of the language at the time of his speaking. Natural language is difficult in this sense because we have to know, for example, to what "values" of X and Y we can apply the transformation rule that takes us from "I never met an X who is taller than Y" to "I never met a taller X than Y." It is clearly not a rule uniformly applicable to uninterpreted strings of symbols.

This difficulty is even deeper than may be at first apparent. For it is not even possible to define the domain of applicability of this rule—and there are many like it—by, say, using lists of male and female nouns, pronouns, and names to suitably amend the rule. Consider a variation of the very example I have cited, namely, the sentence "I never met a smarter man than George." Imagine a detective story in whose first chapter it becomes clear that the sought-after criminal must be someone who is pretending to be something he is not. The master detective unmasks the imposter in Chapter 10, say. The purpose of Chapter 11 is to explain to the reader who has not been able to infer the detective's conclusions from the clues provided throughout the book, just how the detective came to identify the guilty person. In Chapter 11, then, the detective explains that he overheard Mr. Arbothnot make the remark "I never met a smarter man than George" at a literary tea at which the work of the English author George Eliot was being discussed. Mr. Arbothnot had gained an invitation to that tea by persuading his hostess that he was an authority on nineteenth-century English literature. The detective reasoned that anyone who knew anything about English letters would know that George Eliot was a pseudonym for Mary Ann Evans, a lady. Mr. Arbothnot's remark was therefore "ungrammatical," in somewhat the same way that the mathematical expression x/y is ungrammatical whenever $y = 0$; hence Mr. Arbothnot could not be what he claimed to be.

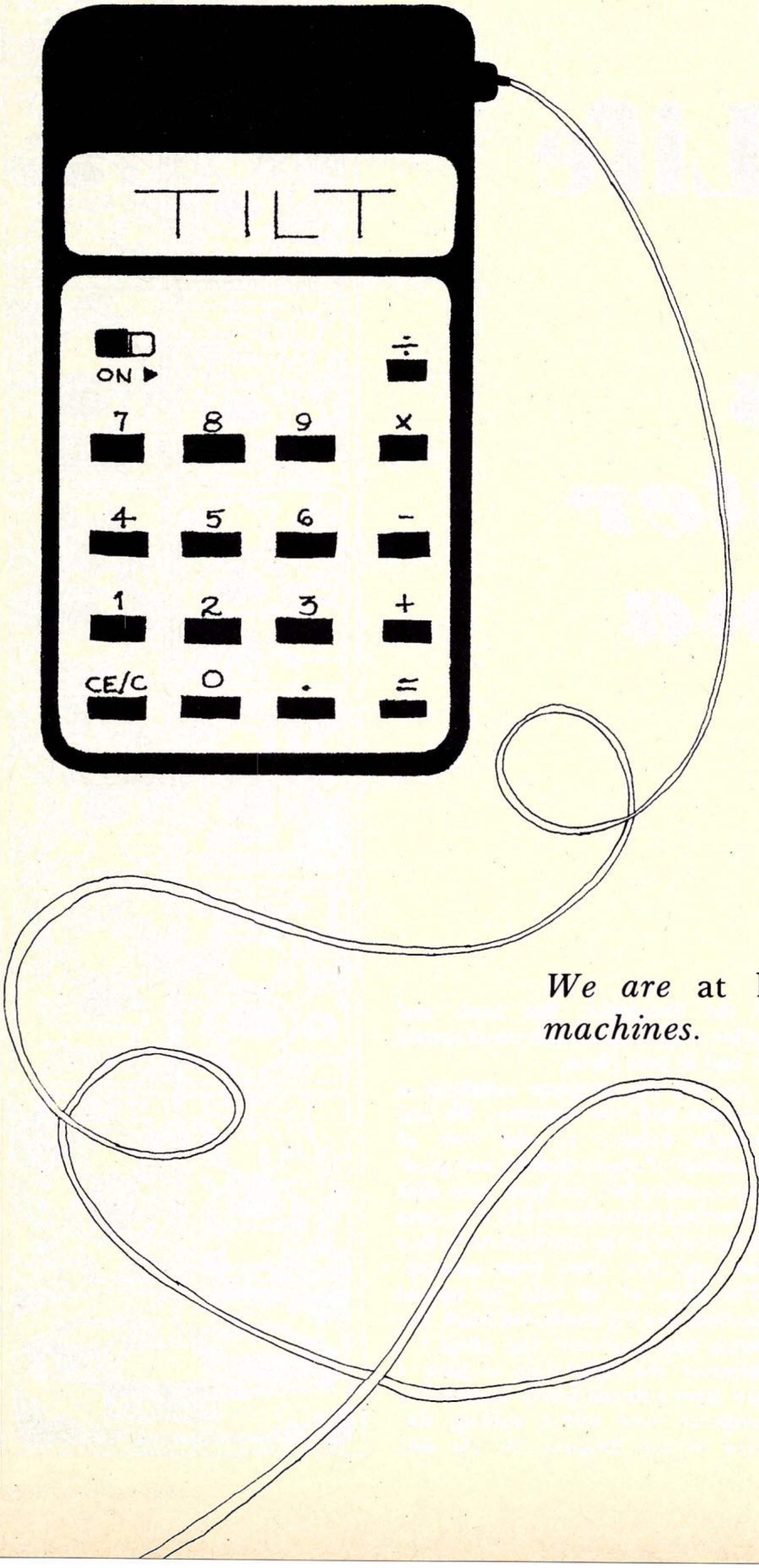
A good detective story, perhaps we should say a "fair" one, is one that gives the reader all the information necessary to discover the truth, e.g., who did it, before explaining how the detective made his deductions. The whole point of a detective story is often just the disambiguation of what one of its characters said in its early parts. The very possibility of spotting an ambiguity, hence of knowing that it requires disambiguation, hence the possibility of solving the mystery, depends on the reader's knowledge of the real world and on the property of natural language that its rules apply to strings of symbols interpreted in real-world contexts. A story of the kind we have discussed cannot be understood in solely formal terms. Interestingly enough, neither can its first chapter be translated into another language without the translator's knowledge and understanding of Mr. Arbothnot's fatal mistake, without, that is, an understanding reading of the denouement provided in the last chapter.

What we can get a computer to do is what we can bring a computer to know.

We will return later to considering the role that context plays in understanding natural language—whether by humans or by machine. For now our concern is still with the narrower question—at least narrower as it is here construed—of the convertibility of human decisionmaking processes into effective procedures, hence into computable processes.

There are, of course, human decisionmaking processes that can be described clearly and unambiguously even in natural language. I have described games here both in natural language and in terms of machine designs that I again described in natural language. Indeed, we could not understand a Turing machine or an effective procedure cast in Turing machine terms, i.e., as a program for some universal Turing machine, without first understanding what it means for one square on a tape to be adjacent to another, what it means to read and write a symbol on a square of tape, what it means for a tape to be moved one square to the right or left, and so on. What is so remarkable is how incredibly few things we must know in order to have access, in principle, to all of mathematics. In ordinary life we give each other directions, i.e., describe procedures to one another, that, although perhaps technically ambiguous in that they are potentially subject to various interpretations, are, for all practical purposes, effective procedures. They rest at bottom on extremely widely shared vocabularies whose elements, when they appear in highly conventionalized contexts, have effectively unique interpretations. Most professional and technical conversations avail themselves of such vocabularies almost exclusively. The problem of converting such procedures into effective procedures in the technical sense, i.e., into programs for Turing machines, is fundamentally one of formalizing the knowledge base that underlies the conventional interpretation of their vocabularies. The more highly standardized these vocabularies are, and the more restricted the context in which they are used, the more likely that this problem can be solved. For, after all, if each symbol of a set of symbols has an effectively unique interpretation in a certain context, and if strings of such symbols are transformed only by rules that themselves arise out of that context, then no question of giving each symbol an interpretation arises in any formal sense at all. A language so constrained is effectively a formal language. Its rules are therefore potentially realizable by a Turing machine.

But then there remain the many decisions we make in daily life for which we cannot describe any decision-



making process in clear language. How do I decide what word to write next? Perhaps our incapability in this respect is due entirely to our failure till now to come to an adequate understanding of human language, the mind, the brain, and symbolic logic. After all, since we can all learn to imitate universal Turing machines, we are by definition universal Turing machines ourselves. That is, we are *at least* universal Turing machines. (Even a physically realized Turing machine is not a mere Turing machine; it may, for example, be a bookend or a paperweight as well.)

We join Michael Polanyi in saying that we know more than we can tell. But in so saying we have come full circle. Our question is, "What can one tell computers?" We have taken telling to mean giving an effective procedure. And the question we are presently entertaining is, "Can anything we may wish to do be described in terms of an effective procedure?" To now assert that there are things we know but cannot tell is not to answer the question but to shift our attention from the concept of *telling,* where until now we have tried to anchor it, to that of *knowing*. We shall see that this is a very proper and a crucially important shift, that the question of what we can get a computer to do is, in the final analysis, the question of what we can bring a computer to know.

For the moment, let us recall that I have already raised that issue; earlier I said that to have a map of a city is not to know the city. Similarly, to be able to tell the rules of chess is not to know chess. The chess master knows more than he can tell. I am not saying here (although I believe this to be true) that we can never find a way to explicate the whole of his knowledge of chess; I say only that we have in this an example of knowledge that is effective even though not presently tellable. Were it true that no chess master's knowledge of chess is fully tellable, would that imply that no computer could ever play master-class chess? Not at all. We shall, as I have said, deal with such questions in the next issues of *ROM*. ▼

We are at least *universal Turing machines.*

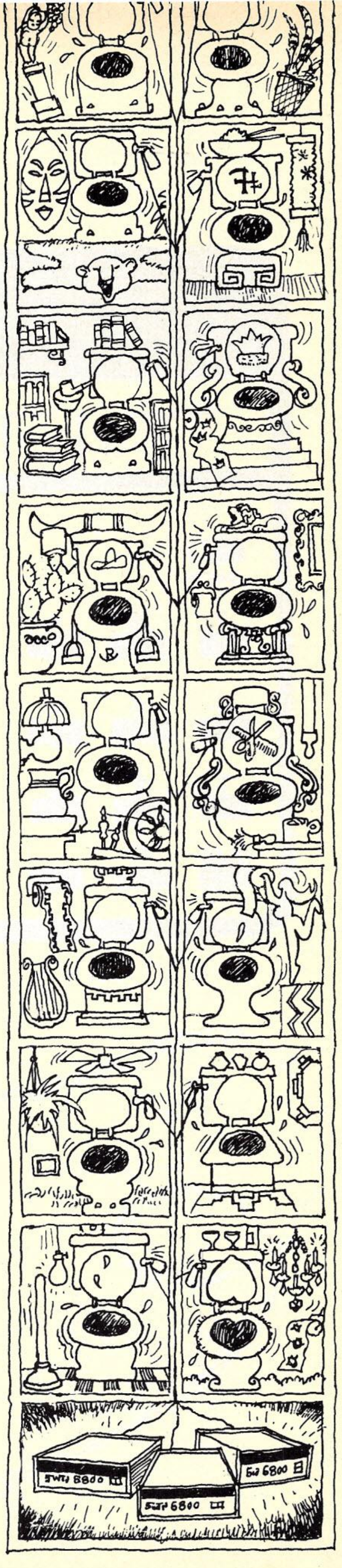
A Day in the Life of *Morsus Computer Taberna* No. 13

by Paul C. Conover

It's 3:30 in the afternoon and the twelve cups of coffee for breakfast were telling me that lunch was late again. I mused that my stomach should surely be used to this routine by now. I'd never gotten out of the store for lunch on time once since I opened.

It usually slowed down around 2:30-3:00 and I let Eddie the Techie (as his Art Crumb psychedelic T-shirt proudly declared) hold the fort while I grabbed a sandwich down the street at the Beanery. The Beanery was my favorite place to eat ever since undergrad days, conveniently located a hop and a skip from the campus. Algorithms, Boolean algebra, and artificial intelligence theory were given at the University; hot food, bad coffee, and essential information could be had at Joe's place.

Joe was the complete antithesis of precision and rote theorems. He had brought balance to the lives of thousands of upper-division and graduate students like me back then with his earthy approaches to life. Put more chili powder in the sauce on rainy days and keep their noses from running. (The Class of '48 held its annual luncheon in a big downtown hotel, but always had Joe cater the affair.) I welcomed the daily visits to Joe's. I may have subconsciously located my computer store within walking distance of the Beanery for the sole



Illustrated by Rex Ruden

He wanted a system of three microcomputers to monitor all the urinals in a twenty-four-story skyscraper.

purpose of keeping a foot on Joe's "steady boat" in the rough seas of retail commerce.

Whatta day it's been . . . . Finally I'm out the door and down to Joe's. When I opened up this morning, Mrs. English was in her car waiting in front again. Right off the bat, same problem as always. She keeps "smoking" her kit. First time out she had her ground and hot wires reversed, second time she had spilled a vase of flowers into it while she was checking a board. This morning it looks like she's melted the front panel right off. It wouldn't have been so bad this morning if she'd just picked up the parts and gone home to try again. Nope, she needed to know how it could have happened so she "won't let that happen again." As it usually does, it took an hour or so of hand-holding and at least half a box of tissues. (She embarrasses easily and cries a lot.) Today I had to tell her that from the looks of the inside, she'd cried on something. (More tears.) It wasn't covered under warranty I explained and offered to assemble it for her.

"No way," she says.

Swell. It seems her psychiatrist told her that she's got to be self-confident and tackle new things on her own. The second time she told me that story I asked her about why she had chosen to build a computer.

"I've never done anything like that before."

Uh-huh.

This morning she's melted the front panel right off.

Joe's place is almost full and I take my spot at the end of the counter near the register. Joe's always down at this end and I'm really fond of the old boy. "Your usual, Nick?" he asks.

"Sure thing, Joe, light on the chili, heavy on the fries." I reply.

While he slaps another patty on the grill, I consider this morning. Some bright-eyed fellow came into the store right after Mrs. English left, bearing up remarkably from the news of catastrophic electric failure. Bright-Eyes, grinning, strode up to the counter with a large roll of blue-line architect's drawings under his arm.

"I'm going to buy one of your seventy-five-dollar computers to automate my house."

He didn't see me raise my hands and eyebrows because his flurry of unfolding plans filled the air with a blinding screen of well-rendered multi-colored dreams and schemes.

"I've read all about how they're making cheap computers the size of my little fingernail," he continued, oblivious to my protests, "You're the professional, you tell me . . . . Which one should I get? The Intel 8080, Motorola's 6800 or the Z-80 I've heard about?"

Before I could explain that maybe he didn't completely understand about microprocessors and how they work in computers, he's pouring over the plans, wanting to know, "Should I put it in the closet here or here?" pointing with a pencil to places on the plan. "Maybe it should be in the garage . . . . Have people put them in the kitchen? What do you think?"

I think he's nuts, but can see how, in five minutes, I can have the man straightened out.

It took a little longer than five minutes.

After explaining that yes, there were microprocessors that were "computers" and then proceeding to the point that the "brain" of a computer *is* the microprocessor chip, I said that the processor, like the brain in our bodies needs a "body" to "haul it around." I pointed out that the processor needs to have memory, I/O cards, power supplies, sensors for inputs, etc. From a quick look at the plans maybe he could get away with his original concept for about forty-five to fifty thousand dollars, not including the waldoes that did the lawn mowing and cleaned the pool.

"Nonsense," Bright-Eyes shot back, "I've read in several magazines that the computers are only seventy-five dollars or so." Clearly thinking that I, the unscrupulous store owner, was trying a "bait-and-switch" scam on him, he continued, "The District Attorney will hear about your false advertising . . . . Crooks shouldn't be allowed to prey on the unsuspecting public."

Uh-huh. Especially the unsuspecting part.

Eddie the Techie was helpful in getting this guy settled down finally. He'd listened to this from the back room while loading some diagnostics; then he came out with an 8080a chip. He tossed it onto the middle of this guy's drawings and told him, "Here's your computer, buddy." Turning from the shelf, he put an assembled 6800 mainframe down and offered a look inside to see how it all fit together.

Bright-Eyes couldn't be bothered with the 6800, he was riveted by the processor chip. "This can't be worth seventy-five dollars."

"Right," quipped Eddie, "That's only about seven bucks."

Bright-Eyes flared up again about false advertising and the phone started ringing. I ducked into the office while Eddie held his hands up like he was cornered by a mugger. One of my suppliers' Shipping Managers was concerned about my order. He had a shipping invoice to send me *one hundred cases* of the *Microcomputer Dictionary*. Did I know that they were packed seventy-two to a case and the freight charges alone would be over a thousand dollars? Could he send them via rail if my store had a siding? (It would save me money that way.) I got the PO number from him and checked my copy. It showed that I'd ordered *twelve copies only* of the *Dictionary* and I told him that. He was sympathetic, but couldn't kill the order himself. He'd put it back through as if I'd refused the shipment and then have someone call me. Fine.

Out front Eddie was laughing hysterically.

Bright-Eyes had decided to put the processor in the living-room closet and had asked Eddie which of these "little metal tongs" should he hook up to the pool heater and which to the air conditioning unit.

Eddie was having an attack.

I ended up selling Bright-Eyes a couple of books on microcomputers and a fist full of magazines and told him that I would invite him to some evening classes I'd be holding soon.

This day was already a record setter. . . .

"Nick?"

"Hmmmmph," I focused on Joe by the register.

"You look tired and worried, my boy," consoled Joe. Joe was always the master of the understatement.

"Joe," wearily nodding my head, "It's the computer store business. I must get three-quarters of the nuts in the world in my shop on a rotating basis."

"Yeah, I know what you mean, Bucko. You know Melvin Schmidt, Class of '44?" Joe asked. "He was in last week with another gadget that he'd invented. Set it right on the counter here and showed me how it worked. Pencils and some Egyptian magic or somethin'."

"Right, I know him, he's in the store every couple of months with his latest 'I'm going to revolutionize the industry' gadget. The last one cost me four hundred dollars," I replied.

"You bought one?" said Joe, incredulously.

"Nope," I countered, "But the demonstration was worth every dollar of it. He was experimenting with pyramid energy, you know—how they sharpen razor blades and the like. His device, about the size of a typewriter with a stainless steel plate on the top, did the opposite, although I don't think that's what he wanted when he started out. First he put a freshly-sharpened pencil on the dish and then he turned it on. While we watched, the pencil got duller and duller. When he turned it off and held up the pencil, the entire point fell off."

"Yeah," said Joe. "He did the same thing here too. But I'd laid my big cleaver down next to the box while all of that was going on and it turned up so dull all you could use it for was to hammer butter. How come it cost you four hundred dollars?"

"He performed this little trick on a display table right next-,to a color CRT. Somehow all the red phosphors were erased," I said. "I had to replace the tube."

Joe brought my lunch and I mused about my wild day. About noontime, the store had filled up with kids from the Engineering and Computer Sciences departments of the University. One of them started making a speech about what a "rip-off" it was for me to be charging seventy-five dollars a copy for 8K BASIC when the duplicating costs couldn't be more than five dollars. When I pointed out that the original code had probably cost a quarter of a million dollars to develop and at seventy-five bucks a copy it was going to take a long time for the software house that had made this investment in product-development to start getting a return on its investment, the kid just told me that I was a capitalist pig. He was going to get some duplicating equipment and start giving away free software. He stormed out of the store spouting something about "freeing the microcomputer 65,536."

Uh-huh.

He probably started writing code about the time he began knocking the beads on his playpen back and forth.

Kids weren't all bad though. Quite the contrary, most were sharp as hell and polite to boot. Some were a real pleasure to have around. Eleven-year-old Sammy and his twelve-year-old pal, Archie, were the future of computing, I was sure. They had come up on their bicycles one afternoon and made a beeline straight for a demo system. Perched in front of the CRT for about an hour and a half playing games and putting the system through its paces, they behaved themselves and never hesitated to relinquish the tube if a paying customer was there to see what's what. Sometimes Sammy even did demos if I was busy with another customer or on the phone.

His father came by one Saturday to ask if Sammy was creating a nuisance by coming to the store every day and "playing with my computers."

I told him no, that Sammy and I had an agreement. Sammy would enlarge my software library of games and little demonstration routines (he wrote them from scratch on the CRT in minutes), and I would let him use the computer all he wanted when I didn't have another customer or demo to run.

His dad said that was fine, same story that Sammy had told him over dinner. We chatted a while longer about Sammy. We determined that he probably started writing code when he started knocking the beads on his playpen back and forth.

Sammy had asked his father to buy a couple of diskettes for storage of some of the programs that Sammy had worked on in the store. He did, and we kept them behind the counter for when the boy came in. Eddie had sneaked a peek at one of them one evening and he told me that there was a Universe war game that looked like it would run on 11K and made *Super Star Trek* look like checkers. Sammy was quite a kid. It hadn't always been so rosy though.

One day the head of the Computer Sciences department at the University had come in and we were talking over a problem he was having. He was telling me that the big time-sharing system at the school was getting so clogged with student jobs and homework, that the lag time between his assignments and the time the students could get the work processed was getting too long for the needs of education. He wanted to know about the University maybe picking up a dozen or so micros to fill in on student assignments. I was telling him how they were just the thing and could see a very large order coming down the pike. He had a need and I could fill it.

Sammy was at the CRT nearby and had overheard our conversation up to this point. He looked up from his successful lunar landing (I think he never missed), and interrupted with a discourse about the sort sub-routines and how they needed to be revised, and how the table-searching techniques needed to be completely rewritten. The fine points were too esoteric for me to follow, but the interruption had cost me the undivided attention of my customer and he'd been more than a little put off by this youngster telling him that his pride and joy number-cruncher was running sloppy programs. He left right away.

I made it clear to Sammy and Archie that it was very rude for them to get between me and a customer (Archie pleaded that he had nothing to do with it), and if it ever happened again, I'd ban them from the store.

A couple of days later the Department head was back to thank me for the help he'd gotten. He wanted personally to thank "my young man" for his excellent advice. Seems that Sammy's suggestions had set the wheels in motion for a careful scrutiny of the sub-routines and some tune-up was in order. Benchmarks now indicated that the through-put was about thirty percent faster on about seventy-five percent of the student workload. The classes were able to get their work in on time and the whole system was greatly improved. He'd also come up with some other ideas on how to use several micros with the course material and was going to recommend my store as "sole-source."

I had made a deal with Sammy on software that afternoon.

I finished lunch and kept doodling on a napkin trying to see what Eddie had been thinking earlier today when he told a fellow who wanted to interface twenty-four automatic pin-setters in his bowling alley to a microcomputer so that he could keep score for his patrons. Eddie had told him that the special interface would cost around twelve hundred dollars. It kept looking to me more like about twenty *thousand.* We wouldn't give a formal quotation until I talked with Eddie, that was for sure.

I'd been burned before. A designer consultant had come to me with a proposal that we work together on a consulting job he'd landed. I said fine and we signed on the project. The customer wanted a system of three microcomputers that would monitor all the urinals in a twenty-four-story skyscraper, flushing them every hour if the door to that lavatory had been opened during the previous hour and not flushing if the door hadn't been opened. Also the system was to stagger the flushes so that the water pressure for the building and three city blocks around wouldn't drop suddenly.

Now this youngster was telling him that his pride and joy number-cruncher was running sloppy programs.

We'd indicated that the system design would be done in ten weeks and delivery of the software and computer hardware could be made just about any time after that. That was nine months and two nervous breakdowns ago. Seems that the only system that could handle the software was an IBM 360/155. Whoops. We're still working on it, but it'll be a major thunk on my bottom line this year.

I'd finished scratching at the napkin and was paying Joe off when Archie came tearing down the street on his bike. "What's the matter, Archie?"

"Golly, Mr. Carey. . . . Sammy needs to know where your order books are. Right after you left a man came in to look at computers and Eddie was busy on the phone when Sammy and I got there. He's from an accounting firm or somethin'. . . . Anyway, Sammy showed him how easy it was to make a computer work. . . . He wrote a program right on the spot for the guy and now wants to get the order for an 8800b with 42K, three floppies, and a CRT. He also wants a graphics printer and Sammy can't find the books."

I beat it back to the store in time to find the gentleman shaking hands with Sammy. Sammy had told him that we could do all the software he needed (for a price) and the president of the largest stock brokerage company in town had just shaken hands with my "Software Manager" on an eighteen thousand dollar equipment order. I was introduced as the fellow to give the money to. I got the check on the spot.

Eddie broke away from a heated discussion with a customer to hand me a phone message. He had that "why me, Lord?" look on his face. The guy he was talking to was demanding his money back on a 16K ROM card kit he'd bought here last week. There weren't any programs on it when he finally got it put together.

The phone message was from the vice president of Major Publishing. He was going to initiate legal action to recover the freight cost for shipping one hundred cases of *Microcomputer Dictionaries* halfway across the country and back again unless my check for the full amount was in his hands Monday morning.

Whatta day!

Just before closing, the brain surgeon who had built his own stereo gear from kits several years back came in for the eleventh or twelfth time. The second time he had brought it back, we'd assembled it for him, after repairing and replacing the scorched parts. It seems that he can't resist opening up the lid and poking around inside to see how it all works. From the cursory glance inside this time it looks like he'll establish an all-time record: his repair bills are now equal to the original equipment cost.

Just about 10:30 p.m., as I was going home for dinner, I got the report from the consulting firm that I'd paid thirty-five hundred to for a survey of my customers and market that would indicate what classes should be offered here to help increase sales and customer confidence. The two biggest areas that computer hobbyists seemed to need training in were soldering and remedial reading.

Uh-huh. It's been another one of those days. ▼

Forget It !

A Short Story by Henry Melton

Carlos Walker had the most thoughtful wife. He told her so while he shook the fancy wrapping paper free from the tiny package she had gotten him for his birthday.

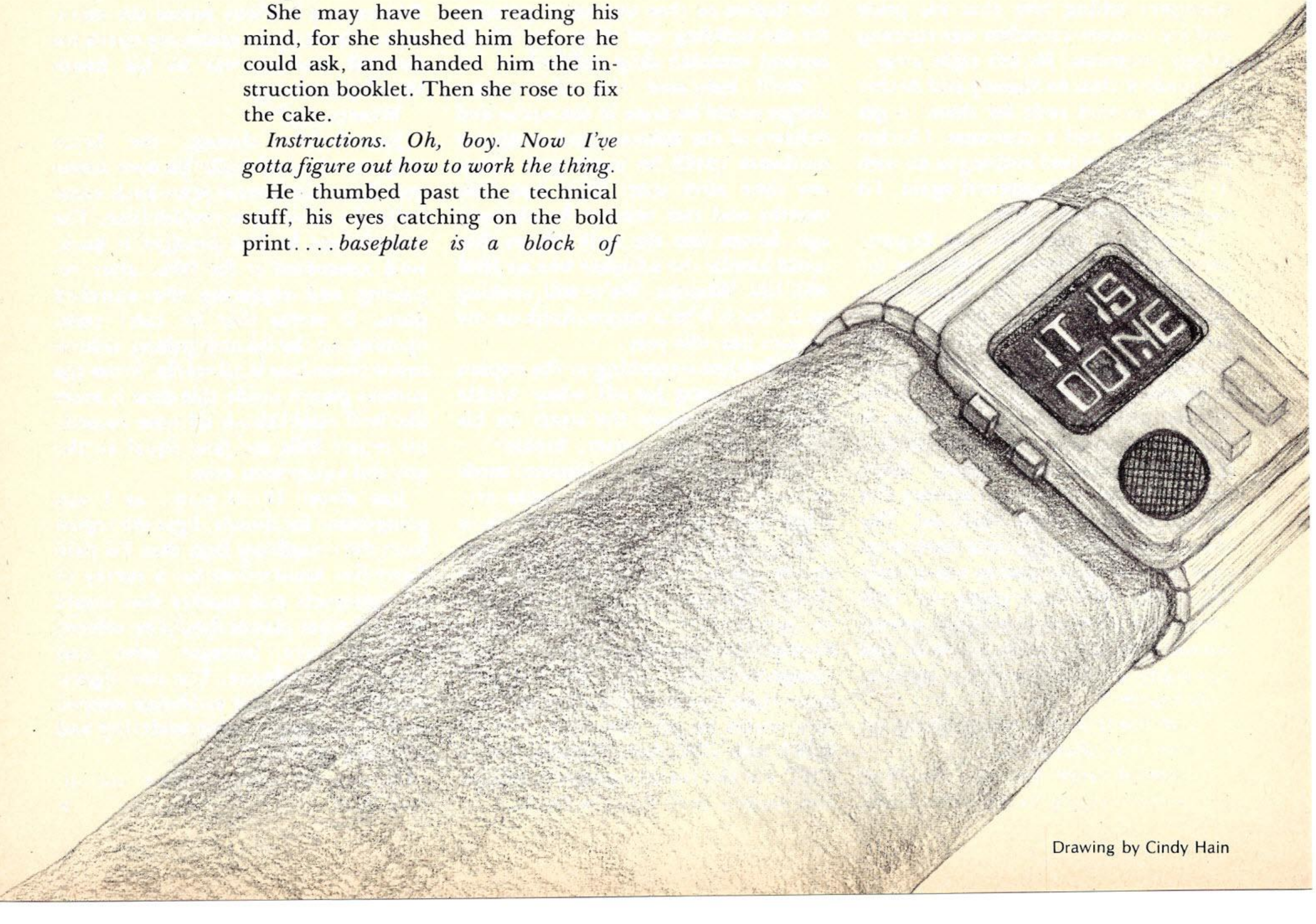
It was a beautiful computer—a gold case on a gold watchband, with an elegant soft black display screen. Deb had been subjected to his wishing aloud for this model since they had hit the market, but he hadn't expected her actually to get him one. It must have done horrible things to her budget.

She may have been reading his mind, for she shushed him before he could ask, and handed him the instruction booklet. Then she rose to fix the cake.

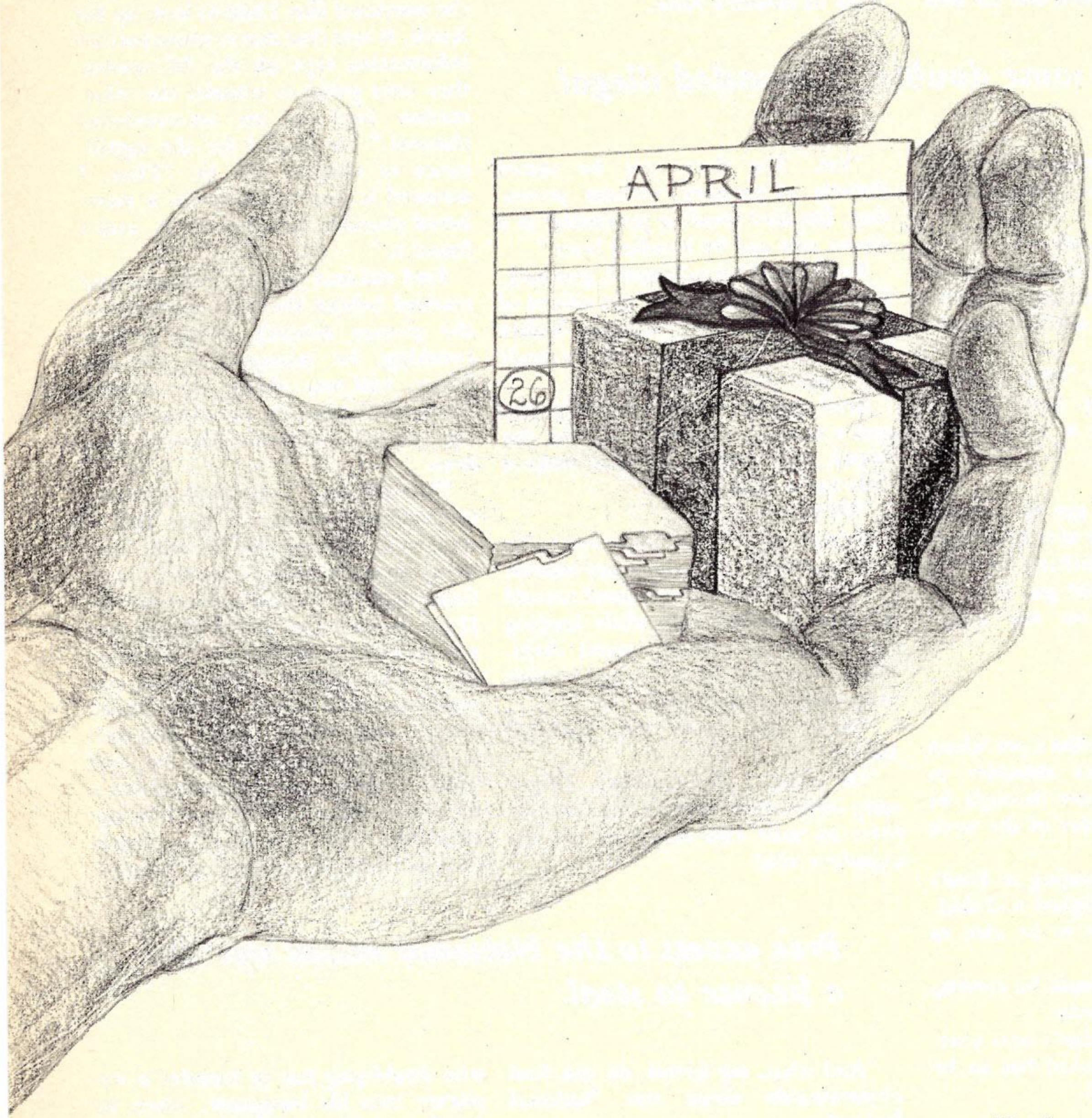
Instructions. Oh, boy. Now I've gotta figure out how to work the thing.

He thumbed past the technical stuff, his eyes catching on the bold print. . . . *baseplate is a block of electron-hole-pair holographic memory with a nearly unlimited capacity. Comptron guarantees that the memory cannot be filled in the lifetime of the original owner. . . . The thin-plate WPU. . . . accepts instructions in an expanding subset of 2012 REVISED U.N. STANDARD ENGLISH. . . .*

The good stuff, the real operating instructions, Carlos found several pages later. He was pleased to see the boldface notice that the contents of the entire booklet, as well as a simple



Drawing by Cindy Hain



It seemed to burn on his wrist all the way up to the office.

instruction course in operating the computer, were available in the computer's own memory at the call of a code word. He scanned the rest of the page which gave the voice-coding procedure, then set it aside.

He brought his arm before his face. "Computer, key to my voice."

"It is done." The computer's voice was male and very much like his own.

He had some doubts. It sounded illegal.

The screen had displayed his own words in red and the computer's in blue.

"Display the index, please."

The little blue lines of data rolled up the screen—tiny, but clearly readable. There were a lot of useful functions listed there.

Finally Carlos said, "Stop. Make a note of Deb's birthday, April 26, and be sure to notify me a week before then."

"It is done."

Carlos nodded and got up, pleased with himself. The odor of rich German chocolate cake was drifting into the room. *Maybe I'd better get it to remind me to stay on my diet—tomorrow.*

* * *

Fred Browell looked like a pet falcon perching over Carlos's shoulder as Carlos put his computer through its paces calculating the day of the week Fred was born on.

There was such a longing in Fred's voice when he finally sighed and said, "What I wouldn't give to be able to afford one of those."

"I hear the prices should be coming down some time next year."

Fred nodded, "But that's next year. The thing I have in mind has to be done now, or not at all."

"What's that?"

Fred looked uncomfortable. Carlos got the strong impression he hadn't meant to bring the subject up. Fred eyed the wrist-computer critically, then asked, "What kind of data-rate can it handle?"

Carlos shrugged and asked it. The display screen filled with technical specifications, while it vocally reviewed the high points. Fred listened carefully to the listing of the audio and visual band widths the device's sensory transducer arrays were capable of handling.

Fred frowned a moment in mental calculation, then asked, "Can you store video-to-memory at that rate?"

There was silence. A smile flickered across Carlos's face, and he repeated the question for Fred. The computer knew its master's voice.

"Yes. The data can be stored directly to memory, without processing. Detailed recall or processing at a slower rate can be handled later."

Fred mumbled to himself, grinning. Carlos could almost see the visions of kilobucks that were dancing before Fred's eyes. Nothing but money made Fred quite so gleeful.

Fred turned to him with a big grin and a slap on the back, "Carlos, my friend, how would you like to make a little spare change?"

"What's the deal?"

Fred glanced down the hallway. No one appeared to be taking any interest in the two of them. Fred eased around to where he could talk while keeping an eye on the people around them. When he spoke, his voice was considerably softer than usual.

"Have you used the National Index?"

"Of course." *Who hasn't. When every information file you need is right there on your Vidi terminal, why go anywhere else?*

Free access to the National Index was like a license to steal.

"And what, my friend, do you find objectionable about our National Index?"

"Oh, not much—other than the constant file charges and those nuisance priority codes." Carlos hated those priority codes. Nothing was worse than knowing that the information you needed existed, just a keystroke away, and then not knowing the code to get at it.

Fred smiled knowingly. "Exactly, my friend. Have you ever paused to think how the information in the Index gets to your Vidi?"

"It's hooked to the Tieline. I suppose they send it out that."

"Right! And do you know how?"

"No."

Fred glanced down the hallway again. "I didn't either," he confessed, "until I ran across a notice in one of the restricted files I had to look up for Kordi. It said that due to unauthorized information taps on the NI system, they were going to transfer the information feed to an *un-numbered* channel." He paused for the significance of that to sink in. "Thus, I assumed it was currently on a *numbered* channel. I looked for it, and I found it."

Fred slid into Carlos's desk seat and reached behind the Vidi terminal for the picture controls. With a little tweaking, he managed to roll the image half-way down the screen. Above the image, on one of the entertainment channels, was a dancing gray area.

"Raw file data. Constantly updating the holding memory in your Vidi. I think it is unscrambled as well. Your wrist-computer said it can store data faster than the Tieline updates the NI. If you ordered it to record what it sees and you left it pointing at a Vidi screen set up like this for a few hours, you could have the whole National Index on your arm. For *no* charge, and with no restricted files. Some of the stuff would gradually get out of date, but a lot of it wouldn't." Fred eyed him significantly. "And I'm sure you might find a friend, he pointed to himself, who would pay you to transfer a duplicate into his computer, when he finally can afford to buy one."

Carlos had to admit that it appealed to him. But he had some doubts. "It sounds illegal."

Fred frowned as he readjusted the Vidi, "I don't think so. The notice I saw implied that the NI people were making the change to comply with some kind of legal ruling. I'd bet it *will* be illegal, after they make the change. Besides, you can always erase the copy if we find out later that it's illegal."

Carlos chewed on his tongue; it was tempting. "Let me think about it."

"Don't think too long. They program the network on the first of the month."

* * *

Carlos did a lot of thinking that night. The way Fred put it made the whole thing sound so easy. But he also remembered how stiff the laws had gotten on copyright matters in the last couple of decades as copying into digital storage systems became so easy. And he only had Fred's word that it would be legal.

In fact, he only had Fred's word on a lot of things. Carlos glanced down at the computer on his wrist. How did he know it was really possible?

"Computer, is it possible for you to store digital data that is displayed in real-time on a standard Vidi channel?"

"Yes."

"Would you be able to index that information later at my request?"

"That would depend on what type of data was recorded and which encoding methods were used. If the data is in a standard code, it would be possible."

Carlos leaned back in his chair, meditating on possibilities. A minute later he abruptly got to his feet and went into his study where he kept his home Vidi. With some experimenting he finally found the way to make the picture roll down as Fred had done on the office machine. But in this case, there was no hidden data.

Wrong channel. I should've noticed what channel that was.

"Computer, what channel were Fred Browell and I watching at about eleven this morning?"

"I do not know."

"Why not? Couldn't you see the channel indicator?"

"I do not know. At that time, there was no command to record, and so I did not do so."

Carlos growled at the thing. There were too many channels to hunt through, hundreds of the things. It would take him all night if he had to hunt it up himself.

There's really only one way to find out. Like he said, I can always erase it later.

Carlos tapped Fred's name and I.D. on the keyboard. The half of Fred's face that was visible on the misadjusted screen lit up when he saw who it was calling him.

Carlos asked, "What channel was that?"

"473."

"How long do you think it would take?"

"I've never seen a file that was more than half a day out of date. Give it twelve hours."

Fifteen minutes later, with a little coaching from the computer to align its camera's field of view with the data on the screen, Carlos left the room. The wrist-computer was propped up facing the Vidi screen, with little blue letters on its face spelling out the message, "Recording."

Twelve hours. Should finish just before I have to leave in the morning. I hope it works.

* * *

It seemed to burn on his wrist all the way up to the office. It seemed forever before he could take a break and talk to it. He found a nook and checked the hallways.

Just like a criminal. This isn't like me.

"Computer, display your answers only."

It printed in tiny blue, "Okay."

"Is the data you recorded off the Vidi in clear code?"

"Yes."

"Can you index files from it on my command?"

"Yes."

"Any file?"

"Yes."

Carlos was nervous like a little boy on his birthday. Free access to the National Index was like a license to steal. "Display the file on current international monetary exchange rates."

Out it came, scrolling up the screen at a rate for comfortable scanning.

"Wait! Back that up to the beginning!"

The image scrolled back down until the very first of the file filled the screen. Carlos read it with a sinking feeling.

"WARNING: Any duplication of this information into any technological storage or display system without a registered authorization by the National Index Corporation is a felony under the Information Ownership Act of 1997."

At tax time, the scanner hunts for that warning notice.

Carlos stared at the warning for a moment, then asked, "Are there more of these warnings in the data?"

"Yes."

"Where?"

"They appear at the beginning of every file, immediately after the file heading."

"Every file?"

"Yes, would you like a count? It will take several seconds."

"Yes."

There was an uncomfortable pause. Carlos wasn't used to computers taking so long to answer. Finally it displayed the number. "43,339,083."

Oh, boy. Was Fred ever wrong on that one! Not only is it illegal, it's forty-three million times illegal.

Carlos pondered over his options for a good five minutes. The National Index was a big prize to give up. But Carlos had a painful honest streak in him.

"Computer, erase the data block you recorded last night off the Vidi screen." *Fred'll hate me for this, but it isn't his computer.*

Carlos glanced down at the little message in blue and pulled himself upright in his seat. The words read, "I cannot erase the data."

"Why not?"

"This computer memory system was not designed to erase any of its contents. The memory block was designed over-large for the job requirement, allowing the expensive selective-erasure function to be deleted. If a new or edited copy of any file is needed, sufficient memory space exists to have both copies. Any named file will be represented by the latest copy. However, earlier editions are always available."

"You can't erase anything?"

"That is correct."

* * *

Carlos argued with his arm for an hour, forgetting that he was supposed to be at work. It just happened to be Fred who went out to look for him.

"There you are, my friend. What have you been up to?" he asked with a smile.

Carlos showed him, and the smile drooped. "Still," argued Fred, "it's good data. I don't know why you would *want* to erase it."

"It's illegal, Fred," Carlos explained, as if Fred was a little on the thick side.

He was going to have to find an acceptable way to break his beautiful little machine.

"Well, you don't have to tell anyone you have it. You don't have to use it if you don't want to."

Carlos shook his head. "No, Fred, you know better than that. This little wrist gadget is duly registered in my name. Its contents are legally available to anyone with the right government form, including the tax people every year. I couldn't keep it secret any longer than a couple of months at the most. I even checked the NI file on its own procedures for tracking down information thieves. Every time a data bank is checked through a privacy scanner, like at tax time, the scanner hunts for that warning notice that I showed you. With forty-three million of them, it could scarcely miss it in my case.

"We've got a problem."

Fred stretched and got to his feet. "Yes, I'd say so, my friend. You've got quite a little problem there. Good luck to you." And he walked off.

Carlos watched him go. Somehow he wasn't surprised.

* * *

For days, Carlos's wrist felt like it was wrapped in lead. The longer he thought about those millions of theft alarms screaming, just waiting for someone with the right machine to hear them, the more it seemed as if he was going to have to find an acceptable way to break his beautiful little machine. Deb wasn't going to like that. *He* wasn't going to like it. Already the thing was indispensable. It was a universal note pad and appointment calendar, reminding him of things he had to do. As absent-minded as he got at times, that function alone was worth almost any price to him.

Almost any price. Certainly not a jail term.

Deb Walker was half puzzled, half gratified that her husband spent so much time studying the instructions for the machine she had given him. He seemed quite faithful in his study. But it was strange he didn't seem to be enjoying it much.

Tax time was approaching and Carlos was much more worried than his wife. He sat in the study, muttering to himself. That was new. He didn't normally talk to himself.

"It's all your fault." He addressed himself to his computer. It didn't answer. It never did to his accusations. "If you could just learn to forget!"

"I can forget."

Carlos's mouth dropped open. "What do you mean?" he growled. "For weeks now, you've told me that you can't erase a thing from your memory—and now you say you can forget?"

"That is correct."

"Which is correct—that you can't erase, or that you can forget?"

"Both are correct."

"Explain that to me, if you please."

"There is no method I can use to erase the memory. As I have told you, only a complete power failure could wipe any data from the memory, and since I am a sealed unit, any attempt to discharge the lifetime power cell would cause irreparable damage.

"However, the memory system I use has no absolute addressing system. All data are relatively addressed from an arbitrarily chosen point in the uniform homogeneous block. This basic reference coordinate is held in a special processor-register. My normal programming cannot affect this register, but the set of Explicit Machine Commands, as listed in the instructions, has the capability of erasing this register. If this is done, the memory contents will not have been erased, but without a method of locating these memories, they will be effectively forgotten."

Carlos nodded to himself as he tried to imagine what it was saying. It couldn't erase—but it could lose the map to part of the memory. He tried, for the hundredth time, to comprehend the sheer magnitude of that memory space on his arm—a solid hologram, with the active elements being individual orbiting electron-hole pairs in that special kind of mathematical space created by crystalline semiconductors. Every word ever written by mankind throughout the ages could be easily expressed in that pattern, and then as easily lost if the writer forgot the key. The National Index would hardly make a ripple. Lose the key, and everything is lost.

"All or nothing, right? How much will you lose?"

"Everything but the Explicit Machine Commands. All of the files that you have set up. All of the routines programmed in at the factory, and all of the initialization."

"Is there a way to save the factory stuff?" Carlos hated to lose those, they created most of the utility of the gadget. He had seen the EMC instructions and he could tell that he wasn't enough of a programmer to be able to do anything useful with them.

"Only if there is another memory block to store the data in."

Carlos glanced at the Vidi on his desk. "Okay, then. Let's get started."

* * *

Tax time came and went, and Carlos breezed through it with a smile. A number of people noticed that he had broken loose from the gloom that had been hounding him. He actually met the day with a smile. Fred looked at him speculatively from time to time, but Carlos always seemed to be late for a meeting when he dropped by for a chat. Everything seemed to be sailing smoothly.

Until one day he came home from work to find his wife in a stormy rage. Carlos *tried* to find out what was wrong, but whatever the sin he had committed, it must have been mortal. He found himself barricaded in the study, up against a formidable wall of angry silence. He didn't understand—until he remembered the date. *April 27. Oops! It was going to be a long night.* ▼

New kid on the block!

But watch out he means business

PERSONAL COMPUTING EXPO

COMES TO NEW YORK FOR BIG BUSINESS

It's a brand new show in the world's biggest economic center specifically for manufacturers and buyers who are into personal computing. For the first time, this booming field will have a New York Coliseum showcase in the major population center in the east. It is planned as the largest public show of its type in the world that will attract enthusiastic buyers from a multi-state area.

WHY NEW YORK?

New York is the economic nerve center of the world. It also is the world's communications focal point, the one place that will put personal computing in a significant spotlight. New York is surrounded in depth by people who work in the computer field, by computer learning centers, universities, personal computing clubs, and thousands of others whose lives are affected by computers.

From this vast potential, Personal Computing Expo will draw the hard-core hobbyist, the interested student, and, because of a highly-publicized program of introductory seminars, those who are attracted and fascinated by computing but have not had exposure to the ways and means of becoming personally involved.

SHOW MANAGEMENT

Personal Computing Expo is being produced by H.A. Bruno & Associates, Inc., a firm in the exposition and promotion fields since 1923. Highly skilled in the production and promotion of consumer and trade shows, the company currently promotes the American Energy Expo, the National Boat Show, Auto Expo/New York. Promotion assistance also is currently rendered to the National Computer Conference and the Triennial IFIPS Congress in Toronto.

The show producer has promoted successful shows in the New York Coliseum every year since the building opened in 1957. Staff personnel are thoroughly familiar with the building, its services, management and labor.

EXCITING SEMINARS FROM "BYTE" MAGAZINE

Personal Computing Expo is endorsed by "Byte" magazine, whose staff is developing an exciting series of seminars and lectures for the exposition.

Visitors to the show will be able to attend these meetings free of charge. They will hear from lecturers such as Louis E. Frenzell and Carl L. Holder. More importantly, visitors will be able to attend meetings aimed at their proficiency levels, from beginner through intermediate and advanced personal computing.

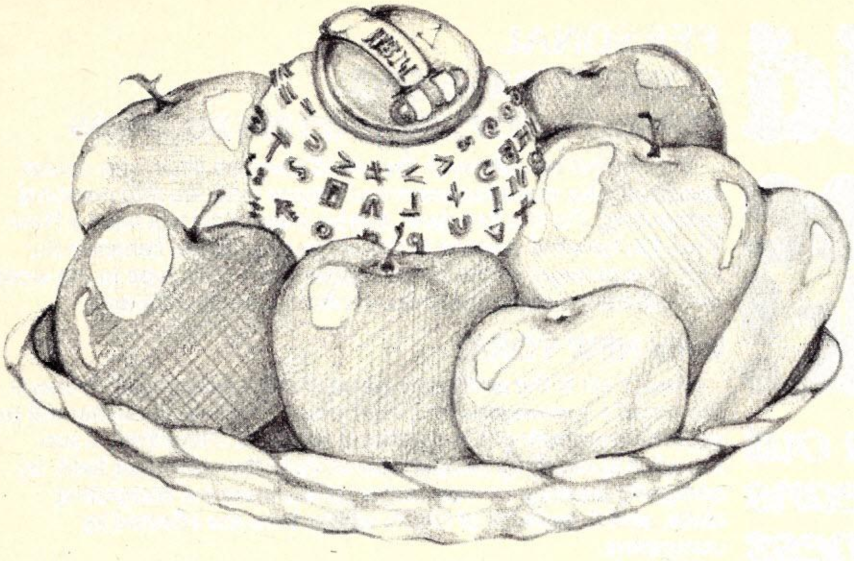
FOR DETAILED INFORMATION CONTACT:

RALPH IANUZZI, Show Manager
H.A. BRUNO & ASSOCIATES, INC.
78 E. 56th Street
New York, N.Y. 10022
(212) 753-4920

Endorsed by BYTE Magazine

OCTOBER 28, 29, 30, 1977

PCE PERSONAL COMPUTING EXPO • NEW YORK COLISEUM



APL.

If you have a small computer, you probably understand the virtues of having some good software to help you program it easily and efficiently—perhaps an assembler and a loader. And if you have an assembler for your computer, you have probably gotten tired of writing out the hundreds of instructions that go into a large program. And if you're like most of us, you have gone on to a BASIC compiler to make your programs simpler, easier to write, and more intelligible. But if you're beginning to find BASIC's loops and tests awkward and tedious, you may not know about the latest rage in programming languages. It's a mathematical language called APL, which is short for (what else?) A Programming Language. Its popularity is such that there are already two journals devoted to it, there are numerous math books published using APL notation, and despite an unconventional character set, many major producers of terminals (including IBM, DEC, Hewlett-Packard, Diablo, Tektronix, Ontel, and Trendata) now make special APL terminals.

Why this interest in a language? One factor is that APL is a very "dense" language—meaning that a great deal can be expressed in a very small space. Because of this, programmers, who typically write about the same number of lines of code per day no matter what language they use, can accomplish much more in APL than they can in BASIC or some of its wordier cousins. This makes it attractive to the companies that hire programmers, and some far-sighted corporations are already using APL for day-to-day work.

Programmers can accomplish much more in APL than they can in BASIC or some of its wordier cousins.

APL was also written to be very consistent, so that a GOTO statement in APL looks like an assignment statement in APL which in turn looks just like a subroutine call in APL, a WRITE statement in APL, or a READ statement in APL. This is because everything that APL can do is expressed as an assignment statement of some kind.

A normal assignment statement in APL looks like this:

```
A←B+C
```

which means the same thing as the BASIC statement

```
LET A=B+C
```

APL doesn't use an equal-sign for assignment because the equal-sign is used for an operation:

```
A←B=C
```

This assigns to A the result of B = C, yielding a one where B = C, and a 0 where $B \neq C$. To output results in APL, instead of assigning the result of an operation to a variable, you can assign

Illustrated by Linda Smythe

o•ma•ni•a

by Eben F. Ostby

it to the symbol ⎕, called "quad." To accept input from the keyboard, you also use quad. The expression

```
    ⎕←12+⎕
```

adds 12 to whatever numbers were entered from the keyboard, and prints the results.

There are over eighty operations defined, and if you don't like those, you can define your own!

The most distinctive aspect of APL is its ability to operate in a parallel fashion—adding a list of twelve numbers to another list of twelve numbers works just like adding 2 + 2.

Simple things can be done in APL in a simple desk-calculator mode, just as in some BASIC systems. To add 2 and 3 in APL, enter

```
    2+3
5
```

The 5 printed by the left margin is the APL system's response, the result of the calculation. To add six pairs of numbers, enter them like this:

```
    2  6  3  1  0  6+4  2  10
33.1  1  0
6  8  13  34.1  1  6
```

Remember that the result of the calculation is what's printed by the left margin.

All the usual operations (multiplication, division, exponentiation, etc.) work the same way, except that the symbols used might differ a bit from the BASIC versions (multiplication is denoted by ×, division by ÷, exponentiation by ⋆). But in APL, there are over eighty operations defined, and if you don't like those, you can define your own! Some of the ones that they give you will sort lists of numbers (called "vectors"), transpose matrices, do least-squares fits, "compress out" elements of a vector, do circular functions, and so forth—all just as easily as adding two vectors.

Suppose you want to sort a vector. In BASIC, the program to do this looks something like this:

```
0010 REM SORT PROGRAM
0020 DIM A(5)
0030 FOR I=1 TO 5
0040 READ A(I)
0050 NEXT I
0060 FOR I=1 TO 4
0070 FOR J=I+1 TO 5
0080 IF(A(I)>=A(J))
     THEN 120
0090 B=A(I)
0100 A(I)=A(J)
0110 A(J)=B
0120 NEXT J
0130 NEXT I
  .
  .
  .
1000 DATA 8,3,5,7,2
```

In APL, no one would even bother

ROMtutorial ROMtutorial

Assembler: A program which converts a program written in mnemonics into the computer's machine language.

Loader: A program which takes a machine-language program and puts it in memory.

BASIC compiler: A program which converts a program written in BASIC into the computer's machine language.

Loops: Segments of a program which are executed a number of times. Usually they consist of a series of instructions followed by a test to see whether the program should execute the preceding instructions, or continue with the instructions that follow.

GOTO statement: An instruction which alters the normal order of execution of instructions in a program.

Assignment statement: An instruction which calculates a function of a variable and puts (assigns) the result to a variable.

Subroutine call: One or more instructions which cause a subroutine (a separate program) to be executed, after which the main program continues its own execution.

WRITE statement: An instruction which causes something (data, text, information) to be printed out on the printer.

READ statement: An instruction which causes data to be input from some source (often a terminal or a mass-storage device, like a disk).

Desk-calculator mode: When the APL system is in desk-calculator mode, it acts like a calculator: numbers and instructions are entered, and the computer responds with the results of the calculations desired.

Matrix: A table of numbers. (Pl: matrices)

Least-squares fit: A mathematical method for approximating a function.

Compress out: Pick out selected elements of a vector.

to write a program to do a sort like this, since it involves only the following:

```
    A←8 3 5 7 2
    A←A[⍋A]
```

APL's functions are extended by the use of operators, too, such as reduction (denoted by /) which applies a function to each element in a vector. For example,

```
    +/1 5 3 3
12
```

adds up the vector 1 5 3 3, yielding 12. Or, for example, the function ⌈ (called "max") returns the larger of its arguments.

```
    3⌈7
7
```

(7 is larger than 3, thus it is the result of the operation and it is printed by the left margin.) Then the "max-reduction" or "max-slash" of a vector yields the largest number in the whole vector:

```
    ⌈/8 3 9 103 2.2
103
```

One of the beauties of APL is that it can operate in a parallel fashion.

All these functions and operators are extended to work on arrays of any rank—which means the array can have any number of rows, columns, planes, levels, and so on—with the restriction that there be no more than sixty-seven dimensions. (That's not a severe restriction: an array of sixty-seven dimensions, with two elements in each dimension, has over 10^{20} elements.)

Other examples of the power of APL abound. Finding the inverse of a matrix is done in one character. Finding the numbers that run down the main diagonal of a matrix can be done in just a few characters—if the matrix is called M, then

```
    1 1⍉M
```

does it.

Usually, printing elaborate business reports involves no looping at all—all the rows of the table to be printed out can be prepared simultaneously. How would you do this? Most likely, you'd define your own function, which, instead of returning the sum or product of its arguments, or sorting them, or whatever, would convert them into the proper form for output. For example, the following function accepts a two-dimensional matrix of numbers, converts them to characters, and labels them on the left with the heading "ITEM N," where N is another argument to the function:

```
    ∇R←N FORMAT MATRIX
[1] R←(((⍴N),5)⍴'ITEM '
      ),6 0⍕N,MATRIX
    ∇
```

The first line, preceded by an upside-down triangle (called "del"), is the header-line; it defines how the function (named "FORMAT") is to be interpreted: it has a left argument (N) and a right argument (MATRIX) and it returns a value (R) whenever it is called. For instance,

```
    123 446 782 FORMAT
3 4⍴⍳12
```

The value returned can be assigned to a variable, or operated on further (for example,

```
    ⎕←'*',A FORMAT B
```

which appends a star to the left of the result and prints it out).

As hardware becomes cheaper and cheaper, small businesses are turning to APL, because programming costs are low, and the finished programs, if sensibly written, are modular and flexible. (The formating program, above, might be called by a program that prints one page of a report, which in turn might be called by a program which assembles the data for the report.)

Recently, I had occasion to work on a general ledger system for a small savings bank in New York state. The completed system kept track of all the bank's ledger accounts, updated them automatically, and produced reports and financial statements. The entire project was completed in just over two man-weeks of work—a similar project in BASIC might have taken several times as long to complete.

With the advent of machines such as IBM's 5100 Portable Computer, small offices are taking advantage of APL. There exist several systems for doctors and other professionals which handle accounts, billing, and insurance claims—all written for IBM's APL machine, the 5100.

All of APL's power comes at a high cost. Its great strength is that it extends it operators to work in a parallel fashion on arrays, and this is ideal for an interpretive language. (Interpreters, because they must "read" each statement of a program before they execute it, usually spend most of their time reading the program, and comparatively little time actually executing it.) But remember that a lot of memory space is required to do operations in APL. For instance, if a program is to create a big matrix (say, 20 by 10 by 10 floating-point numbers), and add a like matrix to it, about 49,000 bytes of memory are required: 16,000 bytes for each of the two matrices, 16,000 bytes for the result, and another 1,000 bytes for control information (addresses of the variables, length and shape information, and various other pointers).

As hardware becomes cheaper and cheaper, small businesses are flocking to APL.

On large APL systems, each user usually gets over 72,000 bytes of memory in his "workspace" (APL's term for the portion of the memory in which the user can store his variables and programs). So, only as the cost of memory has decreased, has APL become inexpensive to run, and it is only recently that it has been available on computers small enough to be in a home or office. Right now, there are a couple of personal computers that support APL; in fact these run APL and little else.

The personal computers usable with APL are, unfortunately, expensive, and, worst of all, they're slow. One, the MCM-800, manufactured by MCM Computers, Inc., is based on the 8080 processor (although it actually doesn't contain an 8080). It is a remarkably complete machine, containing a cassette drive, a keyboard, and a plasma display, all in one box smaller than a portable typewriter. But the software is so complicated that it was taken mostly from the manufacturer's older computer, the MCM-70, and so it doesn't take adequate advantage of the new hardware. (It is, however, much faster than the older MCM-70.) It has a very small memory, too—the MCM-800 is available with up to 16K of memory; to use more, you must resort to a virtual system. Because a virtual memory system relies on software to simulate a larger memory, speed suffers.

A larger APL machine than the MCM-800 is the new IBM 5100, IBM's first venture into the personal computer field. As with everything else IBM sells, it's expensive (ten thousand dollars for the cheapest version). And the 5100 is none too fast, even though it's many times faster than the MCM-70. Its tape system is awkward to use (you cannot update a file and, because of the rigid serial nature of tape format, you cannot read a record from the tape without reading all the records that come before it on the file), and its video screen measures only a tiny five inches diagonally.

But the 5100 does have a very good APL interpreter, and there are a lot of people using it, so it can be assumed that it will be well-supported for some time to come. It has a very good hardcopy printer (a bi-directional dot-matrix machine, which operates at speeds of 80 or 120 characters per second), and it can be interfaced with an extra tape drive or a floppy disk. Curiously, it shares one trait with the MCM-800: it emulates another computer, this one, an IBM-360. If it didn't, it might be twice as fast.

APL is a prime example of a trend in computing: as hardware drops in price, software is developed which takes advantage of efficiencies gained in using lots of hardware. This has kept the price of using APL high for some time—and kept it off personal computers. But as the price of personal computers tumbles and memory capacity explodes, more people may be able to enjoy what APL lends itself to: cheaper software. ▼

ROMtutorial ROMtutorial

Circular functions: Trigonometric functions, such as sine, cosine, tangent.

Argument: A value used as input to a function.

Array: A matrix or table of numbers. An array may also be a list of numbers.

Inverse of a matrix: A matrix which, when multiplied by the original matrix, results in the identity matrix—which has ones down the main diagonal.

Floating-point: One way of storing a number in a computer. A floating-point number includes a value and a scale factor which tells the computer where the decimal point should be located.

Byte: A piece of information consisting of eight bits. It has 256 possible values and is usually used to indicate a typewriter character.

Bi-directional dot-matrix machine: A printer which prints while moving in either direction across the paper, using a dot-matrix printing head, which is a little device with moving pins. Each time a character is formed, the appropriate pins move out to produce the impression. Visualize a football scoreboard, and replace each of the bulbs with a pin, and you have the general idea.

Tape drive: The mechanism which moves, reads from, and writes to a reel or cassette of magnetic tape.

Floppy disk: An information storage device which uses a "disk" which looks like a 45 rpm disk but is coated with a magnetic surface like that used on a recording tape. The disk is flexible, or "floppy." The disk drive rotates the disk and can position a movable head which can magnetically write and read information on the disk.

Plasma display: A technologically-advanced character display. It looks like a bunch of orange dots that glow in different configurations.

A List of Most of the Different Functions in APL

Symbol	Name	Description
*L*ρ*R*	Reshape	Creates a variable of the shape defined by the left argument from the values in the right argument. For instance, `5ρ63 7 ←→ 63 7 63 7 63` .
ρ*R*	Shape	Returns the shape of the argument; e.g., the value that would be used as a left argument for reshape in order to create the variable given as the right argument.
,*R*	Ravel	Turns any variable into a vector (one-dimensional array).
⌽*R*	Reverse	Puts the elements in the argument in reverse order.
L⌽*R*	Rotate	Shifts the elements in a vector around L places.
L,*R*	Catenate	Attach the elements in R to the end of L.
⍉*R*	Transpose	Transposes a matrix along the main diagonal.
L↑*R*	Take	Gives the first L elements of R (or the last L elements if L is negative). If L is greater than ρ R, R is extended with zeroes or blanks.
L↓*R*	Drop	Drops off the first or last L elements of R.
L/*R*	Compress	Returns the elements of R that correspond to 1's in L. For instance, `1 0 1 1 0/'APPLE'` returns `'APL'` .
L\R	Expand	Inserts zeroes or spaces in R where there are zeroes in L. For instance, `1 1 0 1 1 0 1\'ASDFG'` returns `'AS DF G'`.
L[*R*]	Indexing	Works like indexing in BASIC....except that the subscript may be a vector (specifying a number of elements in the argument).
⍳*R*	Iota	First R integers. `⍳5` returns `1 2 3 4 5`
L⍳*R*	Index	The locations of the elements in R in L.
L∊*R*	Membership	Returns a 1 for each element in L that is also in R, a 0 for each element in L that isn't in R.
⍋*R*	Grade up	Returns the permutation (subscripts) that orders R in ascending order.
⍒*R*	Grade down	Returns the permutation that orders R in descending order.
?*R*	Roll	Random numbers in the range 1-R.
⌹*R*	Inverse	The matrix inverse of R (R must be a two-dimensional matrix).
⊥*R*	Decode	Converts R from a number system defined by L into base ten.
⊤*R*	Encode	Converts R from base ten into a number system defined by L.
⍎*R*	Execute	Executes the APL expression represented by the character string R, and returns as its result the result of the expression executed.
⍕*R*	Format	Turns the right argument into its character equivalent.
L+*R*	Plus	
L-*R*	Minus	

by Eben F. Ostby

Symbol	Name	Description
L×R	Times	
L÷R	Divide	
|R	Magnitude	Absolute value of R.
L|R	Residue	For positive numbers, the remainder after L is divided by R.
L⌈R	Maximum	The larger of the two arguments.
L⌊R	Minimum	The smaller of the two arguments.
⌊R	Floor	The largest integer smaller than or equal to the argument.
⌈R	Ceiling	The smallest integer larger than or equal to the argument.
⋆R	Exponential	e to the power R.
L⋆R	Power	L to the power R.
L⍟R	Log	The logarithm of R, base L.
⍟R	Natural log	The logarithm of R, base "e."
○R	Pi × R	R × 3.14159. . . .
L○R	Circular	Circular, hyperbolic, and pythagorean functions: the exact function depends on the left argument. The functions obtainable are: (squareroot $[1\text{-}R^2]$) arcsin, sin, cos, arccos, tan, arctan, sinh, arcsinh, cosh, arcosh, tanh, arctanh, sqrt $(R^2\text{-}1)$, sqrt (R^2+1).
!R	Factorial	
L!R	Binomial	The number of ways that L objects can be taken out of R objects.
~R	Not	Logical negation.
L∧R	And	
L∨R	Or	
L⍲R	Nand	
L⍱R	Nor	
L<R	Less	
L≤R	Less than or equal to	
L=R	Equal to	
L≥R	Greater than or equal to	
L>R	Greater	
L≠R	Not equal	

A Simple Text-Editing System in APL

by Eben F. Ostby

The three APL functions illustrated are an example of a simple text-editing system. Together, they allow someone to input, modify, and output text easily.

The first function, TEXTIN, is for inputting text. When it is executed, the eventual result of the function, called TXT, is set to a character string which is empty—it has no characters at all in it. On the second line of the function, the function waits for input from the terminal—the overstruck quad-quote combination does this—ravels this input (makes sure that the input is a one-dimensional vector), and assigns the result to a variable called IN. Then the program tests the length of IN. If the length is zero, which would mean that the user of the function just hit RETURN on the keyboard without typing any text, then the program branches to line 0. To complicate matters, line 0 doesn't exist—the program line numbers start at 1. So APL interprets this to mean that the program should stop, and the result should be assigned to the variable TXT. If, however, the length of IN is not zero, then the program continues with the next line, which appends IN to the end of the previous input, and finally branches back to the second line to wait for more input.

Thus, when you enter

```
MYTEXT←TEXTIN
```

the program lets you input text until you wish to stop, and assigns the result to MYTEXT.

The second function is called MODIFY. The right argument is TXT, which is text to be modified. The left argument is FRMTO, which tells the program what to modify, and how.

The first character of FRMTO is assigned to the variable DEL; this is the delimiter for the argument FRMTO. On the second line the program removes the first character in FRMTO, searches for the delimiter in the *rest* of the string, and puts the part after the delimiter in the variable TO. On the third line, the part of FRMTO before the delimiter is put back into FRMTO. Then the program looks for the elements of the string FRMTO in TXT. Where they're found, they're replaced by TO, and the result is put in the variable R. When the entire input TXT is exhausted, the program stops, and R is returned to the user.

The third function outputs text. It is given a right argument of the text to be output, and a left argument telling how long and wide the output page is. First, the page number PAGE is set to zero. On the second line, three blank lines are printed, then the page number is formatted and output, and finally an extra blank line is printed. On the fifth line, a line counter is set to zero; this counter is used to determine when the page is full. Then, on the sixth line, a length of characters as wide as the page is taken from the text, the program searches for the last space in the line, and outputs all the characters up to that space. Then the process is repeated until either the page is full, or the text runs out.

If you want to follow for yourself exactly how the programs work, remember that the variable names that follow semicolons in the header-line of the function are for identification of the variables that are used ("locally") within the function. Branch statements are identified by right-pointing arrows at the left of a line. They cause a program to branch to the line number which the expression yields. Finally, remember that the order of execution of functions within a statement always proceeds from right to left, no matter what the functions are—except when parentheses alter this. ▼

EXAMPLE FUNCTIONS IN APL

```
    ∇ TXT←TEXTIN;IN
[1]   TXT←''
[2]  LP:→(0=ρIN←,⍞)/0
[3]   TXT←TXT,' ',IN
[4]   →LP
    ∇


    ∇ R←FRMTO MODIFY TXT;DEL;TO;LOC
[1]   DEL←1↑FRMTO
[2]   TO←(FRMTO⍳DEL)↓¯1↓FRMTO←1↓FRMTO
[3]   FRMTO←(∧\FRMTO≠DEL)/FRMTO
[4]   R←''
[5]  TRANSF:R←R,(LOC←¯1+TXT⍳1↑FRMTO)↑TXT
[6]   →(0=ρTXT←LOC↓TXT)/0
[7]   →(∨/((ρFRMTO)↑TXT)≠FRMTO)/TRANSF
[8]   R←R,TO
[9]   TXT←(ρFRMTO)↓TXT
[10]  →TRANSF
    ∇


    ∇ SIZE OUTPUT TXT;PAGE;LINECT;LINE;WID
[1]   PAGE←0
[2]  NWPG:3 0ρ' '
[3]   ¯42↑⍕PAGE←PAGE+1
[4]   ''
[5]   LINECT←0
[6]  NWLN:LINE←SIZE[2]↑TXT
[7]   ⎕←(WID←1+SIZE[2]-(⌽LINE)⍳' ')↑LINE
[8]   →(0=ρTXT←WID↓TXT)/0
[9]   →(((SIZE[1]-5)=LINECT←LINECT+1)/NWPG),NWLN
    ∇
```

The Noisy Channel

How'd you like a nice little DECwriter for $799? Well, wait till Christmas or so and DEC will probably have a present for you. Seems they're coming out with another generation and old trusty will be relegated to the personal computing small-business user -- at a reduced price, of course.

* * *

Well, word from The Valley has it that the sixteen-bit micros are marching right along. Supposedly coming from a coast manufacturer this fall is a personal computer with a heart from National Semiconductor. Using the IMP-16L developed by the Microcomputer Systems group of National, the as-yet-unnamed computers will leap on the market with a 64K RAM, TTY and video interfaces, as well as, for some strange reason, a card reader. (Anybody still trying to convince the small-business man that cards are the way to go?)

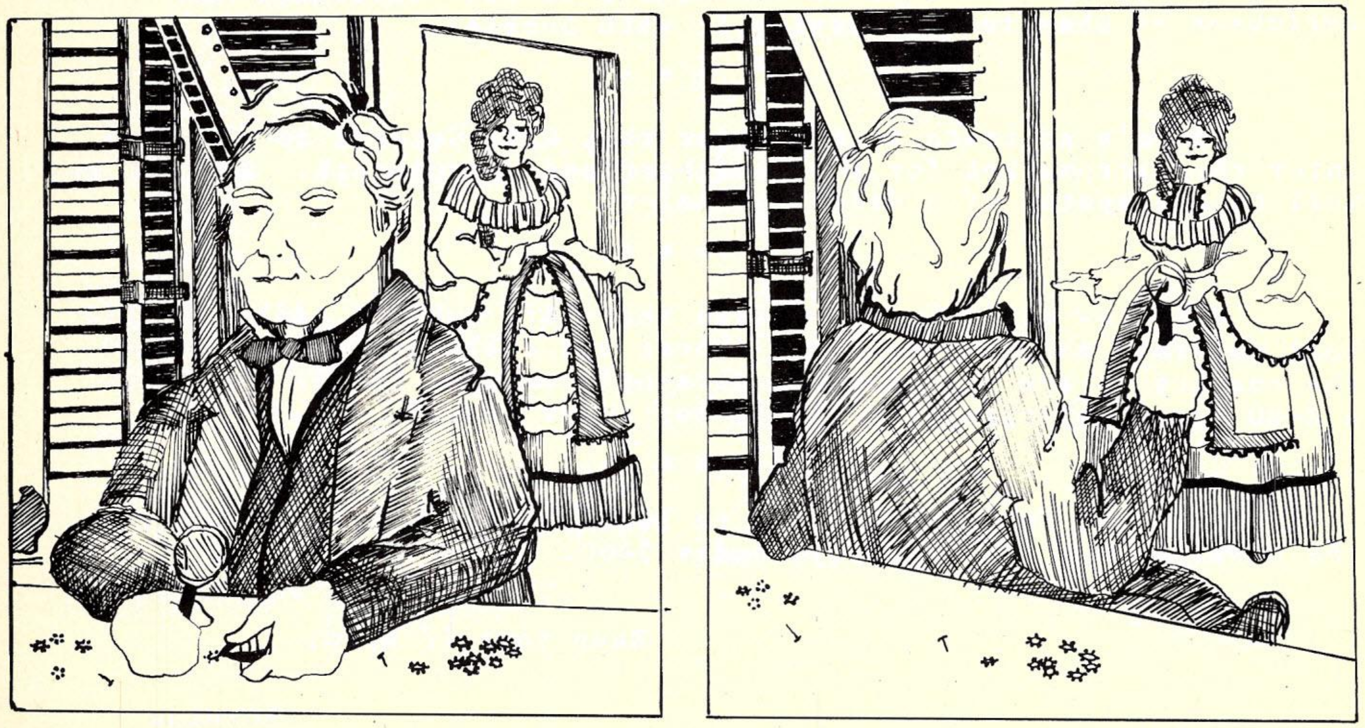
Fast and furious with a typical register-to-register addition time of 4.9 microseconds and memory-to-memory addition of 8.9 microseconds, the unit is starting out with substantial software support, including memory and peripheral diagnostic routines, software DEBUG, assorted cross assemblers, and FORTRAN.

It will be priced for Christmas at $999 complete with removable access panel that crashes the system to prevent unauthorized use. Spare panels anyone?

* * *

Wondering what will happen to the old calculators, now that the computers are here? Well, the story has it Southwest Technical

BABBAGE AND LOVELACE



"Babbage, what are you doing?"

"Microminiaturization, my dear."

Cartoon by Linda Smythe

Products has an umbilical cord coming for you -- the calculator interface plugs into the I/O slot. Although the keyboard and chip add-on are not as fast as the hardware arithmetic units the big machines play with, it allows you to save a lot of memory. Mathematical functions can now be turned into subroutines instead of byte hogging in your assembler's machine code programs.

With a SWTP/Japan having blossomed shortly after the cherries, can SWTP Co. Ltd. be far away? Or how about SWTP S/A? Today Texas, tomorrow the world?

* * *

Parker Brothers in video games? Not now, according to the firm. But then again, they've just released Code Name: Sector, a submarine board game complete with a microprocessor-controlled submarine. Call the sub TMF 0970. Using MOS logic, only 1K of ROM memory, and sixty-four bytes of RAM, it's not making Fairchild quake. But with a name like Monopoly behind them, they could be starting a mind-Bogling Scrabble.

* * *

There's more than one way to export TVs -- call them video monitors, for instance. Although Japan has agreed to cut back on its shipments of boob tubes to the U.S., there's nothing that says they can't modify them for compatriots. Plug it in to your S-100 and you'll still be able to watch Happy Days.

Also from the land-of-the-rising-Sony: it won't be called Betamax II, but it will compute.

* * *

Who's the manufacturer that's designing a symbolic virtual memory that's plug-compatible with the S-100 bus? Can't tell you, but it's rolling into town in six months or so. Curiouser and curiouser -- they're even trying it with bubbles.

* * *

There's no truth to the rumor that Data General is about to enter the micromarket for your neighborhood computernik. But DEC may well be a computer of a different color.

* * *

Bally -- those bouncy people that turn out the addicting pinball machines dotting the candy stores and college campuses across the country -- are entering the personal computer field. Look for a Z-80 based supergraphics system for under $1,000.

* * *

Is Texas Instrument about to launch a 70K bubble memory that the hobbiest can afford? Say, under $400.

Keep your I/ open,

ROMulus

The Brain Robbers

by

Hesh

Wiener

Imagine, if you will, that you are the president of a bank or a large company.

Walk through your data-processing department. Look at the people there. They seem younger and brighter than the clerks behind desks in your other offices. They are dressed modishly; they affect stylish haircuts. Listen to them. They speak a language you cannot understand, although you recognize it as computer jargon.

Inspect your data-processing center, with shadowless lighting and constant air conditioning that makes it impossible to determine whether it is day or night, weekday or holiday. It is a place apart from all others in this world; that much is clear.

Now gather from these places the most alert people, those whose faces speak of achievement made and achievement yet to come. Look at them one by one. Among them are thieves, frauds, and embezzlers. And you cannot determine just who they are.

Some of these people first broke down the defenses of computer systems when they were at school; it is a common enough prank. Others asked the questions you have never dared to ask: Which work is checked and which is not? Which manager watches the figure of his young assistant, ignoring those in the daily reports on his desk? Some of your people have asked these questions, all right, and some have got the answers.

What if there is a thief among your necessarily trusted staff, and what if his larceny is finally detected by a usually inattentive superior? If the crime has persisted for months, will the criminal be turned in—or will your embarassed officer hide the facts, shame defeating duty? This sad ploy has been played out before, and it will surely play again, like one of Shakespeare's lesser works at Stratford or, more accurately, like the legend of Robin Hood.

The Robin Hood syndrome is what criminologists call the mental trick that enables a man of otherwise stringent conscience to rob an institution or let it be robbed. *People* are not hurt, he reasons; the *organization* is impersonal, just a thing. Even the courts have sympathy for this point of view, sending common thieves to prison far more readily than criminals whose offenses were carried out by means of a pencil and paper or an electronic signal.

The fascination for jurists and injured businessmen is with the methods of the thief, a fact which has often brought about a pact of necessity where the criminal goes free in return for giving up the knowledge that made his crime possible. Defense against tomorrow's crook, it is reasoned, is well worth immunity for today's.

Illustrated by Rex Ruden

That price may indeed be fair. Scientists and criminologists who have studied data-processing pirates invariably conclude that the only defense against them is vigilance. But deciding who shall be watched and who shall do the watching is not an easy task. A business cannot afford to wage a constant war against an enemy hidden among its employees.

But fight you must. With each passing day the toll among computer users climbs higher. In a study of nearly four hundred computer crimes, the leading expert on the topic, Donn Parker of the Stanford Research Institute, estimated that the average loss sustained by a victim of computer crime is more than $500,000. Mr. Parker's figures do not include the Equity Funding Corporation insurance fraud, an incredible scheme that resulted in direct losses of $200 million and indirect losses of $1 billion!

The computer-embezzler does far better than his more traditional counterpart. The Federal Bureau of Investigation estimates that the average white-collar crime costs the victim organization $20,000, only one twenty-fifth as much as a typical computer crime.

No one can say with certainty how many computer crimes go undetected. There are cases on record which demonstrate that criminals have pursued their illicit interests for years before being caught. Many will never be nabbed, their crimes buried in the books of their victims like concrete-clad bodies sunk in lakes by old-time racketeers.

While it can safely be assumed that many crimes are discovered by auditors, few will ever be reported to the public. No bank or business wishes to seed insecurity among its customers or stockholders. Frequently the criminals make private arrangements with their victims therefore, which include suitable repentance and restitution. The white-collar criminal is rarely vicious and frequently is more threatened by the shame of publicity than the burden of lengthy penance. He may be able to persuade his employer to view his act as a bad investment, and a bad investment is always treated with the utmost discretion. In a study of automation-crime within the U.S. government, it was estimated that only one crime in five was reported.

Many criminals thus avoid criminal prosecution. Others simply abscond,

CRIME ON THE BANDWAGON

As thousands of people get on the personal computing bandwagon, computer crime may well increase. While there are no reported incidences of such crime yet, it is certain that many hobbyists have already thought about the perfect electronic ripoff. *ROM* discussed the potential of home computing as a training ground for criminals with Stanford Research Institute's Donn Parker. What we learned is that there are compelling reasons for businessmen, large and small, to think about their exposure to data delinquents.

Personal computer users pose mammoth threats to businesses.

Parker told us that it is still too early to make any quantitive predictions, but he thinks that there will soon be a dramatic increase in the number of people with the technical background to commit crimes. At the same time, spreading awareness about computers will make it tougher for con men—or Con Ed, for that matter—to make unreasonable claims about machines and what they can do.

One important group of people to watch out for, we are warned, is the next generation of phone phreaks. A personal computer with a sound synthesizer may be just the right item to wound Ma Bell.

"Computer crime focuses whereever there are people with skills, knowledge, and access," observes Parker.

"The proliferation of personal computers will spread these capabilities among more people. We'll have a larger population with the capability to do bad things."

On the other hand, Parker points out, "personal computing might well change people's attitudes about computers. A better understanding of computers will enable people to tell when they're being properly served and when they're not. Computers will give people a more accurate concept of automation."

In his studies of computer crime, Parker defined several roles a computer can play in a crime. One, in particular, is symbolic: it is when a computer is used to deceive or to intimidate. A positive feature of the personal computer explosion is that hobbyists, with their expanded knowledge of the field, will make this kind of computer crime less successful, Parker believes.

Some not-so-honest computer buffs, on the other hand, envision a deadly game in which they pit their machinery and skills against those of another person or an organization, such as the government or a service bureau. Many firms in the time-sharing business have already thought about that threat and have developed defenses. Their tactics have been sharpened as the number of intelligent terminals has grown. Now the personal computer users could pose the same sort of threat to businesses.

Those institutions with inadequate defenses will find out about their vulnerabilities soon enough, it seems. The most insecure among them are probably the digital communications networks like Western Union's TWX and TELEX, already a common target of phone phreaks. Parker thinks we should examine this area for potential crime by "looking at it from the point of view of the phreaks."

"We're putting a very powerful tool in their hands," he asserts. "They will be able to do more sophisticated things than they have in the past. We're replacing their blue boxes with intelligent blue boxes." All of a sudden it's a scarier world out there.

their crimes awaiting discovery long after they have fled. Escape is facilitated by an obvious characteristic of white-collar crime, and computer crime in particular: the victim is usually the employer of the criminal. The perpetrator, as he formulates his plan, considers the auditors' schedules and searches his victim's procedures for a crucial flaw.

It's the Robin Hood syndrome that enables a man of otherwise stringent conscience to rob an institution.

On the other hand, while some computer criminals reason their escape, others seem to have escaped their reason. In four reported cases, persons have shot computers with guns. In another well-known incident, a clerk at Paris's Orly airport, frustrated by an automated baggage-handling system, removed her high-heeled shoe and smashed in the screen of a CRT. In yet another famous case, a man doused a competitor's computer with gasoline and set it afire.

The average loss in a computer crime is more than $500,000.

The aftermath of a crime may also strain the criminal's grip on sanity. In one case the investigation of a crime produced evidence of surreptitious sexual liasons and, it has been reported, drove one person to suicide and another to an alcoholic's death. When an innocent person was framed, he suffered a nervous breakdown.

The toll, then, also includes losses to the criminal, his family, and friends, which cannot be measured in monetary terms alone. Crime causes losses to society and indirect damage to the victim organization far beyond the amount of the actual theft. No accountant could ever tally the immense cost of computer crime, particularly to the criminal. Nevertheless, the tide is rising, as the growing use of computers provides greater opportunity for mischief.

How, then, can you spot the computer criminal before he strikes—or strikes again? You might examine the profile developed by Stanford's Donn Parker after years of study and interviews with many of the criminals who have been apprehended.

The computer criminal is young; his age is about twenty-five. He is highly skilled, and typically he has risen to a managerial or elevated technical position. He is employed in a position of responsibility by his victim. He may plot with others, but if he does, his conscience may well bother him, as the web of his crime becomes more intricate.

He is cautious. The crime he commits requires only a tiny deviation from his normal pattern of work. He loves a challenge and he may see the computer as a worthy adversary that will yield funds when it is out-witted.

Grading Computers

All 450 seniors at Los Altos High School in California received straight-A report cards this year, which is a great way to graduate. Their brethren at Awalt High in Mountain View, according to the story, weren't quite so lucky—every last one of them flunked.

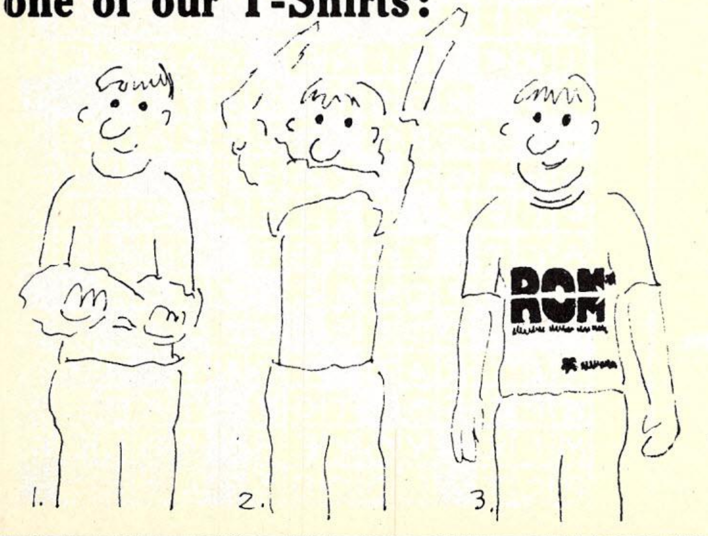
According to Los Altos Principal Robert Madgic, a number of the school district's computer report-card forms were missing. According to the Palo Alto *Times,* the pranksters suggested that more than one student had a personal computer.

The computer criminal impresses his colleagues as bright, reliable, honest, and motivated.

In short, he is precisely the person most likely to succeed at any endeavor.

When he does, you will be out half a million dollars. ▼

This is how to get into one of our T-Shirts:



ROM*
COMPUTER APPLICATIONS FOR LIVING
*** Read Only Magazine**
ROM Publications Corp., Route 97, Hampton, CT 06247

Name
Address
City
State Zip
☐S Qty. ☐Child S Qty.
☐M Qty. ☐Child M Qty.
☐L Qty.
☐XL Qty.
☐Check or money order enclosed $5 ppd.
☐Master Charge ☐BankAmericard 2 for $9 ppd.
Exp. date Card#
Please allow 4 to 6 weeks for delivery.

Advertisers' Index

COLLIER GRAPHIC SERVICES	6, 7
COMPUTER MART OF NEW YORK	47
COMPUTER STORE	46
E & L INSTRUMENTS *Langeler: Mason Inc. Advertising*	31
MIDWEST SCIENTIFIC INSTRUMENTS *Nossaman Advertising*	15
MITS *The Agency*	2
MORE	16
NEWMAN COMPUTER EXCHANGE *Advergraphics*	47
PENINSULA MARKETING	5
PERSONAL COMPUTING EXPO	85
PERSONAL COMPUTING '77	48
PROCESSOR TECHNOLOGY *Bonfield Associates*	52, 53, 54
PROMETHEUS	Back Cover
RAINBOW COMPUTING	46
SOUTHWEST TECHNICAL PRODUCTS *Ken Kirkpatrick Advertising Inc.*	Inside Front Cover
TREASURED COLLECTIONS	Inside Back Cover
XIMEDIA	1

ROM
COMPUTER APPLICATIONS FOR LIVING

For advertising information, please contact the publisher, Erik Sandberg-Diment.

ROM Publications Corp.
Route 97
Hampton, CT 06247
Tel. 203-455-9591

COMING UP IN *ROM*

The 5100 as a Personal Computer
PLATO
A Payroll Program for The Small Business
Computer Wrestling
Measurement, Instrumentation, and The Computer
Charged Couples
The Computerized Greenhouse
How Does a Computer Really Work?
Quality Control with Your Micros
High Speed Printers on a Budget

DON'T FORGET THE DIGITAL FOAM CONTEST

Besides our regular payment to contributors, we will be giving away two prizes for the best applications article on digital foam we receive before January 1, 1978.

First prize is your choice of $500 or its equivalent in Dynacon. Second prize is $250 or its equivalent in Dynacon.

The articles should be 1,500-3,000 words long. They should include diagrams (roughs are fine) and photographs of your system in operation. You must be able to demonstrate that you have it up and running.

For further details see *ROM,* Vol.I, No.1.

Dynacon material kits for experimentation and implementation are available from:

Dynacon Industries Inc.
14 Bisset Drive
West Milford, N.J. 07480

for $10 if you send a check with your order, or $12 if you wish to be billed.

Solution to last month's PROMpuzzle

■	L	O	S	S	■	S	T	A	N	D	■	T	O	M
L	I	B	R	E	■	P	R	I	C	E	■	O	N	E
O	N	I	O	N	■	O	U	R	■	L	A	N	C	E
I	K	E	■	S	I	R	E	■	T	A	R	G	E	T
S	S	■	H	O	S	T	■	R	A	Y	E	■	■	■
■	■	W	O	R	M	■	R	E	P	E	A	T	E	R
A	R	A	B	S	■	C	A	S	E	D	■	A	W	E
F	I	F	O	■	T	R	I	E	S	■	S	L	E	D
A	C	T	■	C	O	A	S	T	■	D	O	E	R	S
R	E	S	P	O	N	S	E	■	K	I	D	S	■	■
■	■	■	O	D	E	S	■	M	E	G	A	■	O	B
T	E	T	R	A	D	■	T	O	N	I	■	A	G	E
E	P	E	E	S	■	T	O	R	■	T	A	B	L	E
M	I	X	■	Y	O	K	E	S	■	A	D	L	E	R
P	C	T	■	L	O	O	S	E	■	L	E	E	S	■

futuROMa

A BYTE AT A TIME

by Bill Etra

Over a year ago IBM announced a bubble memory chip, about an inch in diameter, with a ten megabit (that's one and a quarter million bytes) storage capability. With one and a quarter million bytes you can have one and a quarter million characters on that one-inch disk usable with the Selectric series typewriters. You could put roughly the *New Testament* and all the side books into the Selectric, edit them there, and have them come back out. You could store the whole Library of Congress in the cubic area taken up by about the average size coffee table (if you have an average size coffee table).

The capablility should exist within the next five to ten years for you to store all the knowledge of mankind in the corner of your living room. The big unsolved technical problem is: how do you access all this information? Information without access is knowledge without power, an automobile without gasoline, a camera without film. Functionally it's there, but....

The trouble with bubble memory is that it's accessed serially. This means that each storage cell must be rotated in its entirety to get to any one point. A search through bubble memory is thus serial instead of parallel or random, making access a major stumbling block that must be solved for the full potential of this memory to be realized.

The next stage in bubble memory will probably be the development of independent shifts with complex addressing systems, so the information can be accessed through specialized packets. Presently, bubble memory cannot replace random access memory as the main memory in your computer, because of its slow, serial nature.

What it will replace, in its current state, is disk and tape. What will probably happen in the bubble memory systems we'll be seeing in the next five years is that cassette tape will disappear as a storage medium, disk will be your off-line medium for storing things permanently, and bubble memory will be the on-line medium for exchange with the computer. What we'll have done is to move everything up one notch in terms of storage.

The importance of bubble memory really is in getting rid of the mechanics of the disk. You won't have this mechanical nightmare on the side, which can break, can erase things, can even hit a cigarette ash and dump your memory.

We're approaching the point in memory development where we can store the densest forms of information we have. Remember the computer on *Star Trek?* The cells of that computer, containing all the information, including visual details, were small rectangles which were plugged into the computer. We're approaching this solid state form of storage now with bubble memory. Although the bubble memory is slow, you could put pictures through it, by dumping through a random access memory buffer. Consider a normal full-resolution TV picture. It consists of approximately four million bits, or 500 thousand bytes. To store this is impractical, until we reach the point where we get something like the IBM disk which will store up to one and a quarter million bytes.

Since such a disk is currently so small, and there's hope we can reduce the size of bubble memory even further in the next five years, we're getting close to the point where we will be able to store pictures in tiny solid blocks which can be plugged into the computer at will. Essentially, we're seeing the beginning of the end of mechanical storage devices for home information systems, although the actualization of this event will probably not occur for another twenty years or so. At this point video tape recorders and video disks will be replaced with solid state memory. What will begin to happen soon, within the next ten years, is that you'll see audio equipment—mechanical record players, tape recorders—operating in a digital mode. This will allow for more processing and surer recording for the audio systems as well as greater reliability.

With these very large memories, processing will tend to go into parallel forms. The microprocessors themselves will change from single or double or triple or even quadruple accumulator machines to machines set up in strings and matrices. We'll then be able to process something like 512 by 512 bits of data all at once in a single large matrix. This means that a lot of things will be discovered that were simply overlooked before, because of the tedium of going through the massive records or because of the unavailability of complete data. We can then look forward to another twenty years of scientific discovery merely based on things we knew and couldn't see.

That's part of the thrust we're already in. People are now often merely collating data that's already been collected, to discover what's there (which, of course, is the reason computers have become so important to science). We already had much of the information, but no way of getting through it.

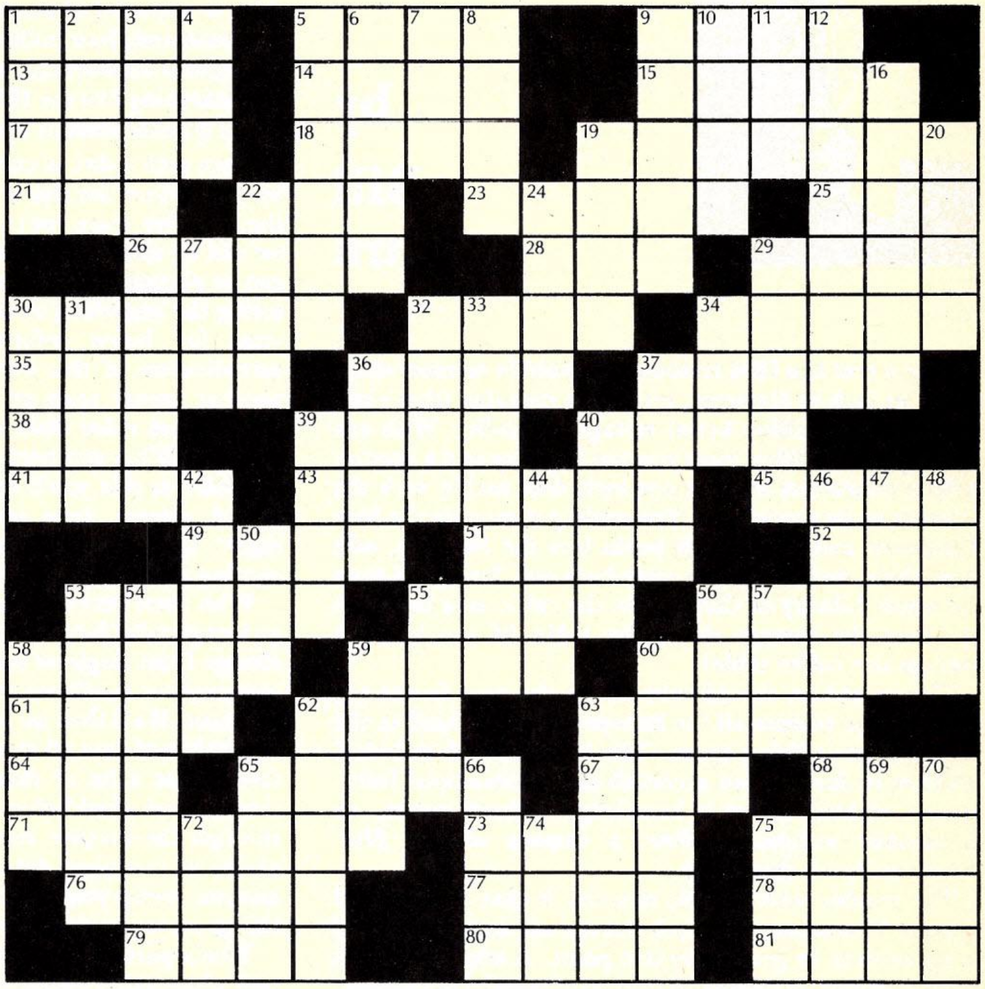
Serial access bubble memory is thus probably only an interim stage of the memory art. What will probably happen is the refinement of other memory forms, for instance holography. In the development stages now, this process uses light reflected off crystals—a combination of liquid crystal or molecular crystal and electron-beam reflection—to scan a material's surface in order to determine the polarization orientation of the crystals, the polarization itself representing the binary storage of information.

Even more intriguing is the fact that it's possible, when polarizing materials, to polarize them at several angles at once. This means that instead of binary storage, you can reach the point where it's feasible to store an *entire byte* at once by having eight angles of polarization at any one crystal point.

With this possible combination of retrieval speed and massive mass storage, we'll be entering a world that even the science fiction writers haven't opened the doors on yet. ▼

PROMpuzzle

by Daniel Alber



The solution to this PROMpuzzle will appear in next month's *ROM*.

ACROSS

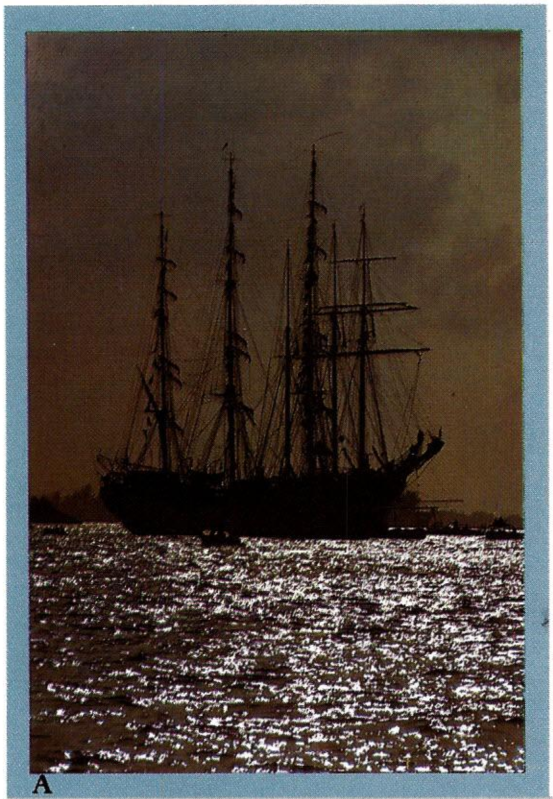
1. ____ package
5. Loafer
9. "When Hector was____" (2 wds.)
13. Type of bean
14. Unit of four conductors
15. Rio ____, Brazil
17. Electrical units
18. ____ Minor
19. ____ ray tube
21. Set
22. Memory-address register
23. Ball player, Roger____
25. Obtain a coded value from a field
26. ____ tower
28. Resistor (abbr.)
29. Gaelic
30. Type of transducer
32. Network (anat.)
34. Uncouth
35. Auto
36. Program executions
37. Fate
38. The E of EOT
39. Season
40. Metallic fabric
41. Coral isle
43. ____ converter
45. Dares, (dial. archaic)
49. Belonging to a young Carter
51. Seep
52. Press service (abbr.)
53. The base
55. Church part
56. The devil
58. Hereditary factors
59. Playthings
60. One of the basic components of a CPU
61. Amo, ____, amat
62. ____ Lizzie
63. Interaction of signals in a data processing system
64. Machine oriented language
65. Heavenly body
67. Special service group
68. Medical suffix
71. Computer memory
73. Breakers
75. ____ commands
76. Where Aswan is
77. Exactly, to____ (2 wds.)
78. Nevada city
79. Observes
80. Remainder
81. Hold tight

DOWN

1. Residue
2. Shade of green
3. ____ modulation
4. Degrees (abbr.)
5. ____ root key
6. Rush
7. Treaty group (abbr.)
8. Cheese
9. Author Nin
10. Strokes
11. Sound of disgust
12. Basic computer instructions
16. Russian town
19. Indian
20. French summers
22. Relocate
24. ____ and crafts
27. Big shot, for short
29. Made a mistake
30. Maple genus
31. Geometric shape
32. Slide ____
33. Unavailabe energy in a computer system
34. ____ devices
36. Cheers
37. Leafy vegetable
39. Mythical river
40. Loll about
42. Decreases intensity in a component
44. Lichen
46. A machine-oriented language
47. Joust
48. A BASIC
50. Management information systems
53. ____ access storage
54. Computers using coded physical quantities
55. Top notch (sl.)
56. Type of lily
57. French friend
58. Legs (sl.)
59. ____ sharing with a program
60. ____, JFET, and CMOS
62. Trading enterprise in South Africa
63. Entices
65. Kennedy or Cod
66. Former Russian ruler
69. Larger than micro
70. On
72. Grain
74. Indian
75. Work unit

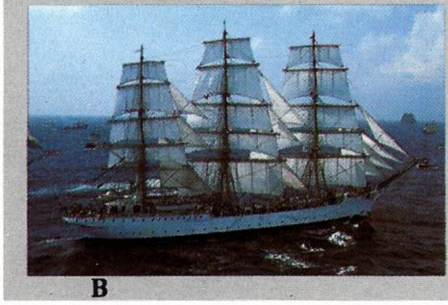
A magnificent first edition . . . pictorial memories of a moment America will never forget . . . or see again

THE SAILING SHIPS

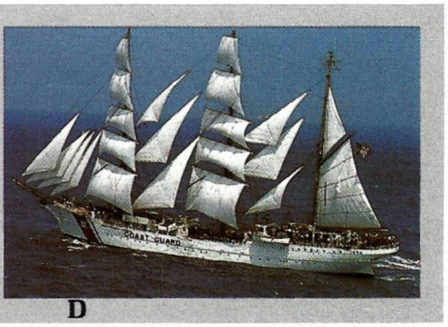

A

Professional quality lithographs of the majestic Bicentennial Ships, now available to the public in an exclusive, limited edition from America's foremost engraver-printer, COLLIER GRAPHICS.

handsomely matted in silvery metal frames; $25 each **unframed, $10 each**


B

C


D

THIS IS A LIMITED EDITION . . . ORDER TODAY WHILE THEY LAST

Never before offered to the public, these magnificent 12x19 full color lithographs of the tall-masted SAILING SHIPS were originally photographed for professional use only!

Now, you can own and enjoy their historic beauty, their incredible clarity, color and reproduction, which gives them almost a three-dimensional quality. And they are only available from COLLIER GRAPHICS — who provide the superb color graphics for America's leading advertising agencies and publishers.

Perfect for your home, boat or office. A lasting gift. Order a set for yourself and one for your children . . . to keep always.

COLLIER GRAPHICS INC., 240 West 40th Street, New York, New York 10018
Please rush the following SAILING SHIPS, securely packaged and postage prepaid, with the understanding that my purchase is UNCONDITIONALLY GUARANTEED by you if the order is returned within 15 days of delivery:
Send me the following print(s). Quantity ________ Price ________ Total ________
A. Christian Radich ________
B. Danmark ________
C. Kruzenshtern ________
D. Eagle ________
Please add $2.50 for shipping and handling for framed print(s) or $1.00 for unframed print(s).
(N.Y. State residents add applicable tax, NYC residents add 8%. Allow 6 weeks for delivery.)
Name ________
Address ________
City ________ State ________ Zip ________
Enclosed, check or money order for $________
Or charge my credit card ________Master Charge ________BankAmericard
Credit Card # ________ Inter Bank # ________ Expiration Date ________
Signature X ________

PROJECT
PROMETHEUS